AVRADCOM

Technical Report -76-0927-F

SINGLE SITE ACTIVATION LOGIC AND DISPLAY

R. LYON
FAIRCHILD CAMERA & INSTRUMENT CORPORATION
SYOSSET, NEW YORK 11791

September 1978

Final Report for Period 30 June 1976 – 30 April 1977

DISTRIBUTION STATEMENT: Approved for public release; distribution unlimited.

Research and Development Technical Report
Aviation Research and Development Command

NOTICES

Disclaimers

Disposition

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS

UNCLASSIFIED (U)

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AVRADCOM-TR-76-0927-F	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) Single Site Activation Logic and Display		5. TYPE OF REPORT & PERIOD COVERED (9) Final Rept. 30 Jun 76 to 30 Apr 77
		6. PERFORMING ORG. REPORT NUMBER (14) ED-AX-99
7. AUTHOR(s) (10) R. Lyon		8. CONTRACT OR GRANT NUMBER(s) (15) DAAB07-76-C-0927
9. PERFORMING ORGANIZATION NAME AND ADDRESS Fairchild Camera & Instrument Corp. Syosset, LI, New York		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62202A 1F2 62202 AH 85 12 34
11. CONTROLLING OFFICE NAME AND ADDRESS US Army Avionics R&D Activity AVRADCOM Fort Monmouth, NJ 07703		12. REPORT DATE SEPTEMBER 1978
		13. NUMBER OF PAGES 93
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (16) 1F262202AH85 (17) 12		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

(11) Sep 78

(12) 104p.

Approved for Public Release

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

(18) USAAVRADCOM (19) TR-76-0927-F

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Charge Coupled Device	Scanning
Single Site Activation	Displays
Wire Detection	Microprocessor
Pattern Recognition	Simulation
	Programming

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The concept of Single Site Activation relative to the one dimensional characteristic of wire/wire like objects was exploited to develop and implement an algorithm capable of providing pattern recognition for this class of object. A FORTRAN program was developed to demonstrate both detection logic and predictive logic which provides a recognition capability in less than a complete scan frame. The continuities requirements as a function of background noise were explored. —> next page

DD FORM 1473, 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED (U)

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

433 500

cont'd A simulation system consisting of microprocessor based hardware was programmed and tested. The simulator provides the capability to generate a 50 element 70 scan scene with a variable background. A "wire" can be placed anywhere in the scene. The continuity criteria can be selected and the success of the pattern recognition algorithm demonstrate by the computor wire detection program.

LEVEL II

FOREWORD

This final report was prepared by Fairchild Imaging Systems, a Division of Fairchild Camera and Instrument Corporation at Syosset, New York under Contract No. DAAB07-76-C-0927. The work was performed under the direction of the Avionics Research and Development Activity, Ft. Monmouth, New Jersey. The program manager for the Army was Mr. A. Kleider.

LIST OF FIGURES

1.0 INTRODUCTION

This final report on the U.S. Army Electronics Command Single Site Activation Logic and Display (SSALAD) Contract No. DAAB07-76-C-0927 summarizes the accomplishments of this project, in which computer processing algorithms for wire detection were developed and implemented.

The SSALAD study presented in this report is based on the feasibility of "single site activation" i.e., the responsivity of a charge coupled device (CCD) detector array to wire-like objects within a single picture element site. An earlier study established the feasibility of single site activation in a CCD wire object detection technique.[1]

Two modes of wire pattern recognition were simulated in computer programs with one of the modes selected for implementation into a test hardware configuration. The simulated modes were "contiguous scan" and "non-contiguous scan" programs evaluated on an IBM 360 computer. The contiguous recognition mode, where successive scans sampled contiguous sectors of a scene, was implemented in the test hardware using an Intel 8080 microprocessor system.

Section 2 of this report defines the basic elements of wire detection and the detection statistics required by the wire determination criterion.

---

[1] Kleehammer, R., "Wire Object Detection Study" Research and Development Technical Report, ECOM-76-0881-F, April 1978

Section 3 describes the computer program for the algorithm using the wire detection assumptions that successive scans are contiguous and wire segments are straight lines. This algorithm was developed and extensively studied on a large scale general purpose digital computer. The implementation of this algorithm in a test hardware micro-computer developmental system is described in Section 4.

Section 5 describes the predictive logic algorithm developed for the more realistic situation of using both a non-contiguous scanning mode and wires that follow a catenary-type path.

A summary of results and the conclusions derived from this study are presented in Section 6. Detailed program listings and flow charts are presented in the Appendices.

2.0 WIRE DETECTION

This section defines the basic algorithm input and develops the statistics of detection information and the relationships between the probabilities of single site detection and the probability of false alarm detection.

The properties of a wire as a one dimensional, extended target and the concept of achieving greater information content from this type of "regular" shape compared to other shapes has been investigated in earlier wire obstacle detection studies[2].

The use of CCD arrays to achieve single site activation was also investigated in an ECCM study[3]. These concepts, verified by experimental data, have led to the present study which defines algorithms for contiguous and non-contiguous scanning of a wire.

In Figure 2-1 is shown the geometric relationships between the wire, the array, and the scanning mode imaged at the focal plane of a CCD detection system. The wire is much smaller in one dimension than the pixel size and is extended to full pixel size in the other dimension. Contiguous scanning of the wire is represented by samples taken in the $S_1, S_2, S_3 \ldots$ mode. Non-contiguous scanning is shown in the spaced S_1, S_{10}, $S_{19} \ldots$ sequence. The output signal of the wire "hits" (which are the input signals to the digital processing system) are represented by [010] where "1" represents the wire signal and the "0" on either side of the "1" represents an absence of signal surrounding all wire-like objects.

---

[2]Kleider, A. "An Experimental Evaluation of Gated Low Light Level TV (GL^{3}TV) For Wire Obstacle Detection", Research and Developement Technical Report, ECOM-4321, May 1975

[3]Kleehammer, op.cit.

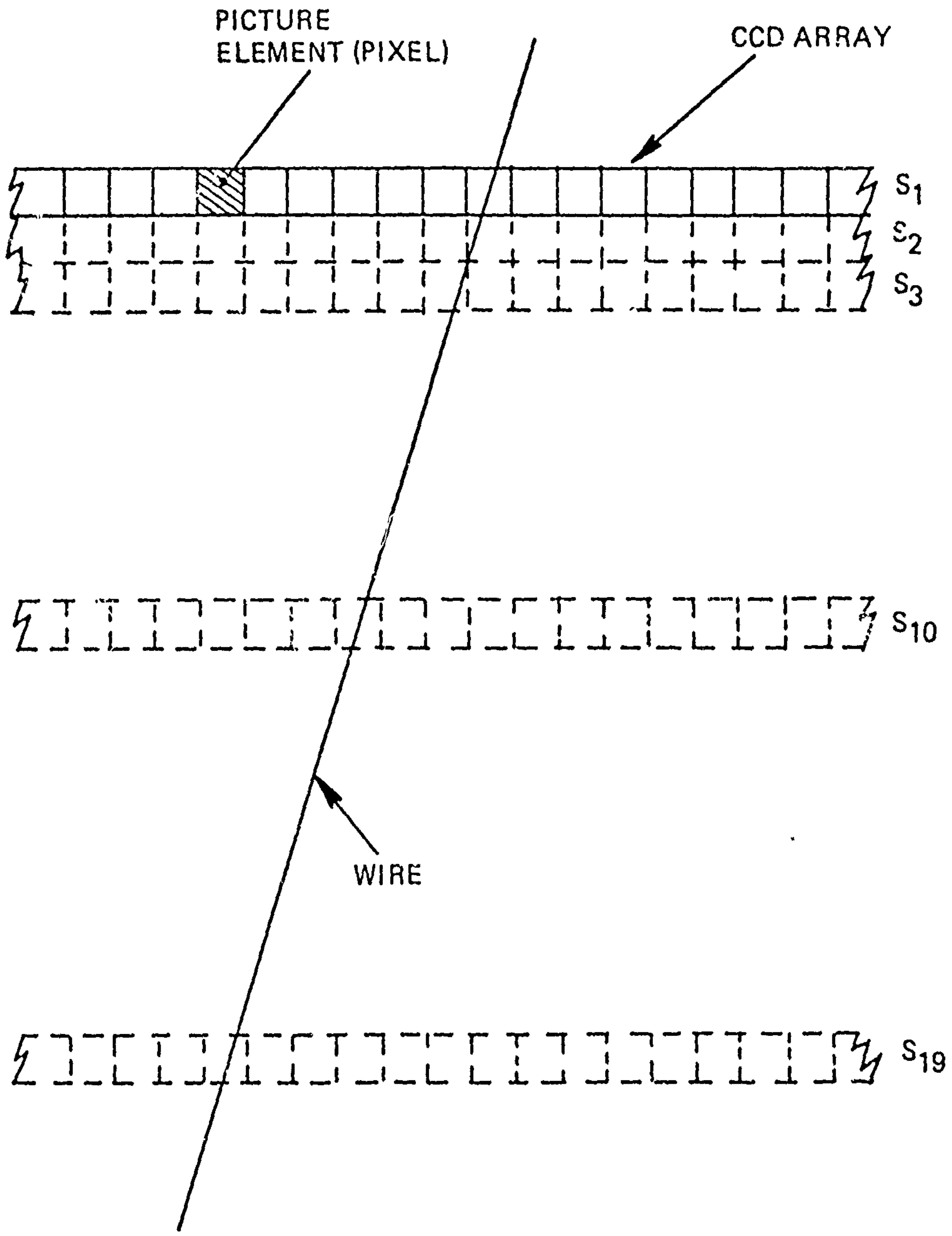


FIGURE 2-1. GEOMETRIC RELATIONSHIPS BETWEEN WIRE, CCD ARRAY, PIXELS AND SCAN MODE.

2.1 STATISTICS

For this discussion it is assumed that the input to the processor is a matrix of binary "ones" and "zeros" representing the detection of events in the "scene". Pixel locations, where the energy density from the "scene" is higher than some predetermined value, produce a "1" condition; otherwise a "0" condition. Due to the presence of a threshold and the binary decision, the "image" with its associated grey levels is replaced by a unit contrast "image" with zero dynamic range. Therefore, it can be assumed that each pixel input is characterized by ;

- its state; 1 or 0
- its minimum probability of detection, Pd
- its maximum false alarm, Pfa
- its location in an array.

Here it is assumed that the signals are represented by the Poisson probability density function as appropriate for many types of optical detectors. Given a mean value of signal from the target wire, Nt, and from the background, Nb, the mean value is still Poisson. Therefore, the probability of detection can be expressed by

$$Pd=P(N,TH) = \frac{\sum_{x=TH}^{\infty} \frac{(N)^x e^{-N}}{N!}}{\sum_{x=0}^{\infty} \frac{(N)^x e^{-N}}{N!}} \qquad (2.1)$$

where $N = Nt + Nb$

TH is equal to the threshold yielding the cumulative probability of detection. The value, TH, must be determined numerically. Once set (in equivalent electrons) TH will determine the reliability of the detection. Further, when the background level Nb is known, TH will determine the value of N that must be received from the target. The probability of false alarm, Pfa, is now defined by the probability of detecting noise alone (Nb) when the threshold is set. Thus, the probability of false alarm is;

$$Pfa=P(Nb,TH)=\frac{\sum_{X=TH}^{\infty}\frac{(Nb)^{X}e^{-Nb}}{(Nb)!}}{\sum_{X=0}^{\infty}\frac{(Nb)^{X}e^{-Nb}}{(Nb)!}} \qquad (2.2)$$

For a given signal level (Nb+Nt) the threshold is chosen according to a suitable criterion. The consequence of not detecting a wire-like object when it is present is very serious so that the Pd must be set as high as possible while allowing some tolerable background level. The set of parameters Pd and Pfa (with threshold value implicit) defines the single site statistics. This pair is a joint probability condition which can be shown to determine the required detection signal-to-noise ratio.

A related factor is the probability of false dismissal, Pfd, which is

$$Pfd = (1 - Pd)$$

This factor must be made as small as possible.

2.2 DEFINITIONS AND ASSUMPTIONS

In a wire detection system the stream of binary inputs to the processor must be searched for configurations of isolated "ones" which indicate a wire. Specifically in the following it will be assumed that the data stream is organized into a raster and that a wire will consist of a string of isolated "ones" in a curve basically described as a catenary (y - a cosh (x/a)). However, it is realized that the wire "hits" can take on a large number of shapes due to optical blurring target signature effects such as glint from surface or water droplets, the angle of the wire, the steepness of the wire curve, and the probabilities that the wire crosses the intersection of cells providing droplets or multiple hits. Secondly, the wire may be long or short. For these reasons the number of steps required is large and a precise and thorough knowledge of the feature extraction algorithm is necessary. An M x L array of pixels will be search for a short line segment. The length of the short line segment is strictly a function of the Pd in each single site. For example, if the minimum length of a wire segment is 7 pixels long and the single site Pd = .9986, the probability of detecting a wire when present is $(Pd)^7 = .990$. The conclusion is that the threshold must be chosen as low as possible to guarantee high confidence in detecting a short line segment and yet be consistent with the Pfa.

Due to various optical effects the single site hits may appear in contiguous (touching) or proximate (near but not touching) patterns. Consider initially the following case:

Hits are contiguous
The wire segment is a straight line
W contiguous hits are required to detect a wire.

These assumptions appear to be reasonable for the detection of short line segments.

2.3 ESTIMATE OF SPURIOUS DETECTION

Once the pattern that would be interpreted as a wire-like object is defined, it is possible to estimate how often false alarms only would randomly occur in such a pattern. This estimate, nf, is given as

$$nf = (a)(b)(c)(Pfa)^W$$

where a = Number of configurations of a W-long line segment that are contiguous and linear,

b = Number of ways of counting a W-long line segment in an M x L array in one direction only,

c = Number of rotations that must be explored to determine if a wire-like object is present,

$(Pfa)^W$ is the probability of occurence of a W-long line segment due to false alarms only.

Estimates of these factors are as follows:

Configurations:

From the above definition of a line segment the line must occur in a 2 x W array where either site in each 2 x 1 must be a "1". Then the number of such configurations is $2^W/2$ because at least half of the elements in a 2xW array must be condition "1". Therefore,

$$a = 2^{W-1}$$

Displacements:

Consider an L x L sub-array of the M x L array. The number of ways a W-long line segment can be

counted in the L x L array is

$$(L)(L + 1 - W)$$

in the parallel direction only and, for the entire M x L array we have

$$b = \left(\frac{M}{L}\right) \quad (L) \quad (L + 1 - W) \; ;$$

Parallel only.

Rotations:

Consider one pixel in the middle of an L x L array. The number of angles that can be defined is simply the circumference of the array divided by the minimum definable increment along the circumference. Therefore

$$c = \frac{4\ L}{(L\ \theta min)} = \frac{4L}{L \tan^{-1}\left(\frac{1}{L}\right)} = 4L$$

Therefore, the final expression is

$$nf = 2^{W-1}\ (4LM)\ \ (L + 1-W)\ \ (Pfa)^{W}$$

Now we assume that W = L and find an expression when the length of the line segment is equal to the number of scans made by an M x L array. Therefore, we have:

$$nf = 2^{L-1}\ (4LM)\ \ (Pfa)^{L}$$

This is a simple expression that has important consequences. Figure 2-2 shows the plot of false alarm, Pfa, versus the number of scans required to detect the object with given false alarm.

This figure is plotted for two array lengths, 1728 and 500 elements. As shown, the number of elements do not dramatically effect the results.

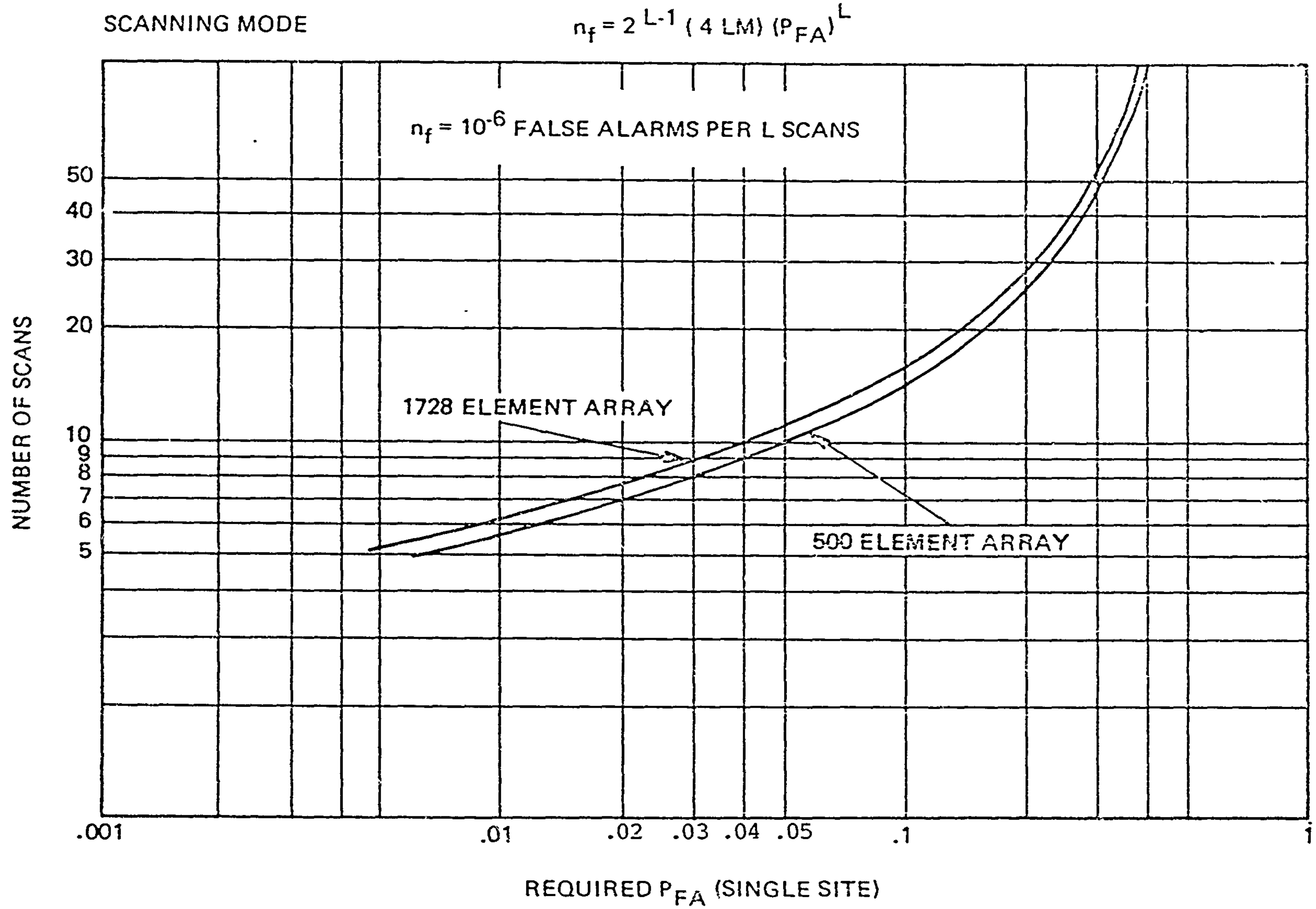


FIGURE 2-2 NUMBER OF SCANS REQUIRED vs P_{FA}

76-350

As expected, the algorithm is more efficient in discriminating against "spurious wires" when the initial noise is low. On the other hand, as the noise becomes very high the burden upon the algorithm becomes severe and in the extreme may not converge at all.

For the 500 element array shown in the figure, the wire can be detected in 7 scans with an overall probability of detection of 0.99 provided that the single site false alarm rate is less than 2%. When this condition is achieved the resulting algorithm false alarm is 10^{-6} per scan as required by the Statement of Work.

2.4 NON-CONTIGUOUS SCAN CONSIDERATIONS

The previous sections have dealt with wire-detection procedures in which successive scans were assumed to be contiguous, such that a wire which was detected as a single site at element j on scan n could be expected to be detected on scan n+1 at element j-1, j, or j+1. If there are gaps in the array coverage, however, the wire-detection procedure becomes more complex. In the initial continuation of a wire from scan n to scan n+1, the single site position in scan n+1 may fall anywhere within a relatively large window, the size of which depends on the length of the scan gap and the maximum allowable wire angle. Once 2 scans have indicated a possible wire, based on the detection and discrimination of single site activation occurences and their locations, its position on the next scan can be more accurately predicted. This predicted position must consider not only the previous slope of the possible wire, but also the basic curvature of a catenary-type wire-obstacle. Thus two different procedures are required for searching the raster for the presence of wires. In the initial operation, the raster is broadly searched for the

presence of consecutive "hits". The angle covered must be large enough to guarantee finding a wire when present. The window size required for this operation is called the "search window". Once two wire hits are found, a linear curve fit can be utilized to predict the location in the next scan where a hit should occur if a wire is actually present. However, the location of a real hit may deviate from a straight line due to various effects. Therefore, a "predictive window" is defined which accounts for these effects. Both of these windows are characterized in the following paragraphs.

2.4.1 Search Window

The size of the search window depends on the length of the scan gap and the maximum allowable wire angle. The scan gap, R, in units of pixels is given by:

$$R = \frac{\emptyset FOV}{\emptyset \cdot N}$$

Where $\emptyset FOV$ = Total field of view

$\emptyset$ = Angular Resolution

N = Number of scan lines

For example, if the number of scan lines is 160 across a FOV of 15 degrees and an angular resolution of 0.1 MRAD, each scan corresponds to a skip of about 16 pixels.

The "search window" size, P, that must be tested to find the next occurance of a wire is given by:

$$P = \pm\ 2R \tan \frac{\emptyset}{2}$$

R = Pixels per skip

$\emptyset$ = Total search angle

For the above example, the "search window" required is ±16 pixels when a total search angle of 90 degrees is used.

2.4.2 Predictive Window

Once two "hits" are found, the mean angle of the wire can be calculated. This information can be used to predict the position on the next scan where a wire "hit" should be located. Consequently the window size may be shortened. This procedure can be repeated for all potential wire "hits" beyond the second. The mean angle of the wire can be normalized out by a simple linear curve fit.

In order to be effective, this scheme must compensate for the inherent curvature of a free hanging wire. Superimposed upon the mean angle of the wire as defined above, the wire may also show an instantaneous slope that causes its actual position to deviate from the straight line predicted above. Thus, the window must be increased in the upward direction by the amount, p+, given by:

$$P+ = R \tan \theta_s$$

Where:

R = number of pixels in step from scan to scan

θ_s = instantaneous angle of steepest slope of the wire.

When the window increase is computed for a scan-to-scan gap of 16 pixels and an angle of 15 degrees, the window must be increased by 4 pixels.

Real wires are likely to be hanging on their own weight and, therefore, will be convex upward. Any deviations from straightness will cause the wire to always appear above the position predicted on the basis of a linear fit and never below. The only reason why the predictive window need be increased in the downward direction is to account for optical effects. Lumping these factors together the window is increased by 6 pixels which includes four above and two below the expected position.

3.0 CONTIGUOUS SCAN PROGRAM

The initial computer program is based on the assumption of contiguous scans. The basic algorithm, program input and output are detailed in the following sections.

3.1 CONTIGUOUS SCAN ALGORITHM

An algorithm was develcped to locate a string of connected single sites within an array which represents the response of linear elements over successive scans. In this program, the scans were assumed to be contiguous and the single sites, representing the wire obstacle, form approximately straight lines. The program isolates and tracks these sites. A 'wire-determination' is made when the number of consecutive sites reaches a given input criterion.

Consecutive scans are considered to be contiguous. This implies that if a wire has been detected at element j on scan n, then on scan n+1, the wire would be detected at element j-1, j, or j+1. Furthermore, although the single site activation logic defines a wire-like object as the binary series 010, provision has been made to accept the series 0110 as a continuation of a wire. This condition is not expected except for quite close range, but has been included in the program for completeness. Thus, any of the following combinations would be accepted as a possible wire-continuation.

scan n	010	010	010	010	010
scan n+1	010	0100	0010	0110	0110

In addition, dropouts may occur due to noise and the wire determination must continue beyond such dropouts. The following cases, in which a single site is activated on the nth and n+2nd scan (but not the n+1st), are examples of 'follow through' a dropout.

scan n	010	010	010	010	010
scan n+1	000	000	000	000	000
scan n+2	0010	0100	010	0110	0110

The program allows 2 consecutive dropout scans before terminating a possible wire continuity.

The normal program mode inputs a percentage value which determines background noise as a percentage of the number of elements in the array. This is accomplished by using a uniformly distributed pseudo-random number generator, and comparing its output (a floating point value between 0.0 and 1.0) with the input percentage. For example, when the percentage of system background noise is input at 1.5, any element for which the random number generator output is less than 0.015, becomes a background noise single site (i.e., a false alarm).

In the processing of each scan, the entire length of the array is initialized to zero, the background noise sites are determined and a 1 is placed in each of these positions. The 'true' sites (i.e., the wire-obstacles) are then overlaid. In the process of overlaying these true sites, the actual position of the wire may remain the same as in the previous scan or it may move up one element position or down one element position; it may also become a double site with a 1 in the same element position as on the previous scan and a 1 in the prior or subsequent array element position; the

site may also become a temporary dropout, disappearing on this scan and picked up again on the next one. The probabilities of the occurrences of these various conditions are input values and the actual determination in each case is made by random numbers. This allows both straight lines and various amounts of wavyness in the wire-obstacle path.

The program builds a table of all single sites which occur from one scan to the next. This table of continuities contains the most recent element position, the number of successive occurrences, and the last scan on which a site was found. The criterion for a continuity requires that the site on a particular scan must have previously occurred at the same element or may have moved one position in either direction. A double site is accepted if either of its 2 elements had been previously isolated as a single site. A continuity is dropped from the table if more than two successive scans have elapsed without an additional entry. That is, 2 successive dropouts may occur and the continuity will be retained (except at the very beginning where 2 consecutive scans are required); on the 3rd successive dropout, the continuity will be eliminated from the table. The criterion for determining that a continuity is indeed a wire-like obstacle is an input value and any wire-continuity which exceeds this value is printed. Because of the developmental aspects of the program, continuities are continually tracked through all scans and the printout occurs only when the continuity is terminated or the last scan has been processed.

3.2 PROGRAM INPUT

Input to the program has been designed as the minimal amount which will allow great variability in all parameters. Thus

basic parameters such as the number of elements in the array and the number of scans to be processed are input values. The background noise and the location of 'true' wire-like obstacles are normally input as percentages of the array length and are computed using a uniformly distributed random number generator. It is possible, however, to input a specific background pattern and/or the initial location of the wire-like obstacles. It is also possible to input a specific series of single sites (which may or may not define a wire) when a specific hypothesis concerning criteria, dropouts, etc., is to be investigated. The wire determination criteria, i.e., the number of scans (excluding dropouts) through which a possible wire is tracked prior to the wire determination decision is also an input parameter. The behavior of the true wires (whether they were input as specific values or computed as random numbers) is controlled by a series of input percentages which randomly determine which of the possible patterns or dropout will occur on the n+1st scan.

Specific input cards are as follows: (All integers are input right justified in 5 column fields, all floating point numbers have 10 column fields.)

1. 1 card (format 20A4) containing the
   RUN ID   Any alphanumeric data in columns 1-80.
2. 1 card (format 7I5, 2F10.0) containing the following run parameters:

Field	Name	Data
1-5	IX	Random number initializer; any odd integer.
6-10	NARRAY	Number of elements in array, NARRAY $\leq$ 1728

Field	Name	Data
11-15	NTRUE	Number of sites to be input by cards. NTRUE ≤50. If NTURE = 0, true sites will be generated by random numbers.
16-20	NSCAN	Number of scans, NSCANS≤100.
21-25	NCNTPR	Wire obstacle determination criteria for printing. i.e., a continuity will be printed if the number of occurrences > NCNTPR
26-30	NARRPR	Single site map option. If NARRPR = 0, the entire array of elements for each scan is printed. If NARRPR ≠0, no printout occurs.
31-35	NDRAW	Extra line option. If NDRAW ≠ 0, an additional line of single site data will be read (see item 5). If NDRAW = 0, no additional single site input is to be read.

Field	Name	Data
36-45	PBACK	The percentage of background sites to be generated by random numbers. If PBACK ≤0, the background will be read from cards.
46-55	PTRUE	The percentage of true sites to be generated by random numbers. (Applicable only when NTRUE = 0).

3. 1 card (format 6F10.0) containing the parameters which control the movement or 'wavyness' of the single site lines. Movement is by random number generation under control of the following parameters:

Field	Name	Data
1-10	PSAME	% of time the site will be the same as on the previous scan.
11-20	PUP	% of time the site element number will be increased by one.
21-30	PDOWN	% of time the site element number will be decreased by one.

Field	Name	Data
31-40	PPUP	% of time the site will be represented as a double occurence, moving up; i.e., a one occurs in the same position and also in the next highest position.
41-50	PPDOWN	% of time the site will be represented as a double occurrence, moving down; i.e., a one occurs in the same position and also in the next lowest position.
51-60	PDROP	% of time in which the site will be temporarily omitted on each scan.

These percentages should obviously add to 100, but the program does not perform this test.

4. For NTRUE > 0 only.
   1 or more cards (format 16I5) containing the initial positions of the NTRUE sites. The program will test the validity of these input values and eliminate any which do not meet the single site criteria.

5. For NDRAW ≠ 0 only.
   2 cards, (each format 16I5).
   This option is used to insert single sites in predetermined locations. The locations would normally be contiguous, thereby having the effect of inserting a short line into the element-scan map at a particular

predetermined position.

The 1st card contains the scans into which the sites are to be placed.
The 2nd card contains the element positions, corresponding to the scans on the first card.

6. For PBACK = 0 only.
The background array (a series of ones and zeroes) for each scan will be read from cards. Thus for each scan, 1 or more cards will be read (format 80I1) containing any single sites desired in the background. This option is not recommended unless NARRAY ≤80 (i.e., 1 card/scan), and NSCAN is small.

If it is desired to have no background sites, use a very small positive value for PBACK (for example 0.0001).

3.3 PROGRAM OUTPUT

Figure 3-1 shows the initial printout for a particular set of data processed in the contiguous scan program. The column on the top left shows that the array contains 180 elements, 100 scans were processed, 2 'true' single sites were input; (thus the percentage of random true sites equals zero); the percentage of random background sites is 2.1; the occurrence of 6 or more connected single sites is the wire-determination criterion; the random number initializer (any 5 digit odd number) is printed only for reference in case the same particular set of conditions is to be rerun.

```
CONTIGUOUS SCAN WIRE DETECTION PROGRAM

   180 = LENGTH OF ARRAY                    PERCENTAGE OF TRUE SITE MOVEMENT
   100 = NUMBER OF SCANS                          75.0  NO MOVEMENT
     2 = SITES TO BE INPUT                        10.0  MOVE UP 1
   0.0 = PERCENTAGE RANDOM TRUE SITES              0.0  MOVE DOWN 1
   2.1 = PERCENTAGE RANDOM BACKGROUND             10.0  DOUBLE-UP
     6 = CONTINUITY CRITERIA                       0.0  DOUBLE-DOWN
 11111 = RANDOM NUMBER INITIALIZER                 5.0  DROP-OUT

COMPUTED OR INPUT SSA LOCATIONS     TOTAL =   2
   15   75

EXTRA INPUT LINE
  SCAN      40  41  42  44  45  46  48  51  52  53  55  58  59  61  63  65
  ELEMENT   30  30  31  31  31  32  33  33  33  33  33  33  33  32  31  30

      CONTINUITIES

ELEMENT     TOTAL     LAST
 NUMBER   OCCURANCES  SCAN

   31         18       68
   82         84       92
   27         98      100
```

FIGURE 3-1. PRINTOUT FROM CONTIGUOUS SCAN WIRE DETECTION PROGRAM

The column on the top right shows the manner in which the true input sites will move across the page for successive scans. 75% of the time the sites will appear at the same element in scan n+1 that it did in scan n; 10% of the time the site will move up 1 element (i.e., the element number increases by 1); 10% of the time the site will appear as a double with the extra 1 at the higher element number (this prints as only 1 single site with the X on the same element as the previous scan); 5% of the time a dropout will occur. These values can be confirmed by examining Figure 3-2. The next output lines show the original true single sites which were input. Below this is the optional input line which may be placed at the operator's discretion. The 2 input sites and optional input line can be seen in Figure 3-2.

The continuity table shows the results of the algorithm wire tracking, which are confirmed in Figure 3-2.

- A wire-like obstacle terminated at element 31 in scan 68 after 18 occurrences. This is the optional input line and careful examination shows that 2 random sites fell (in scans 49 and 68) such that they were included as part of the wire.
- A second wire-like obstacle terminated at element 82 in scan 92 after 84 occurrences. This wire illustrates the general pattern controlling the movement and dropouts along the wire from scan to scan. It also illustrates one of the many problems introduced by the background. The last single site listed in the continuity is at element 82, scan 92, whereas the wire obviously continues. This problem results from a combination of the wire moving on 2 successive scans and random sites falling such that a single site is not found on scan 93. Only single sites are printed in Figure 3-2.

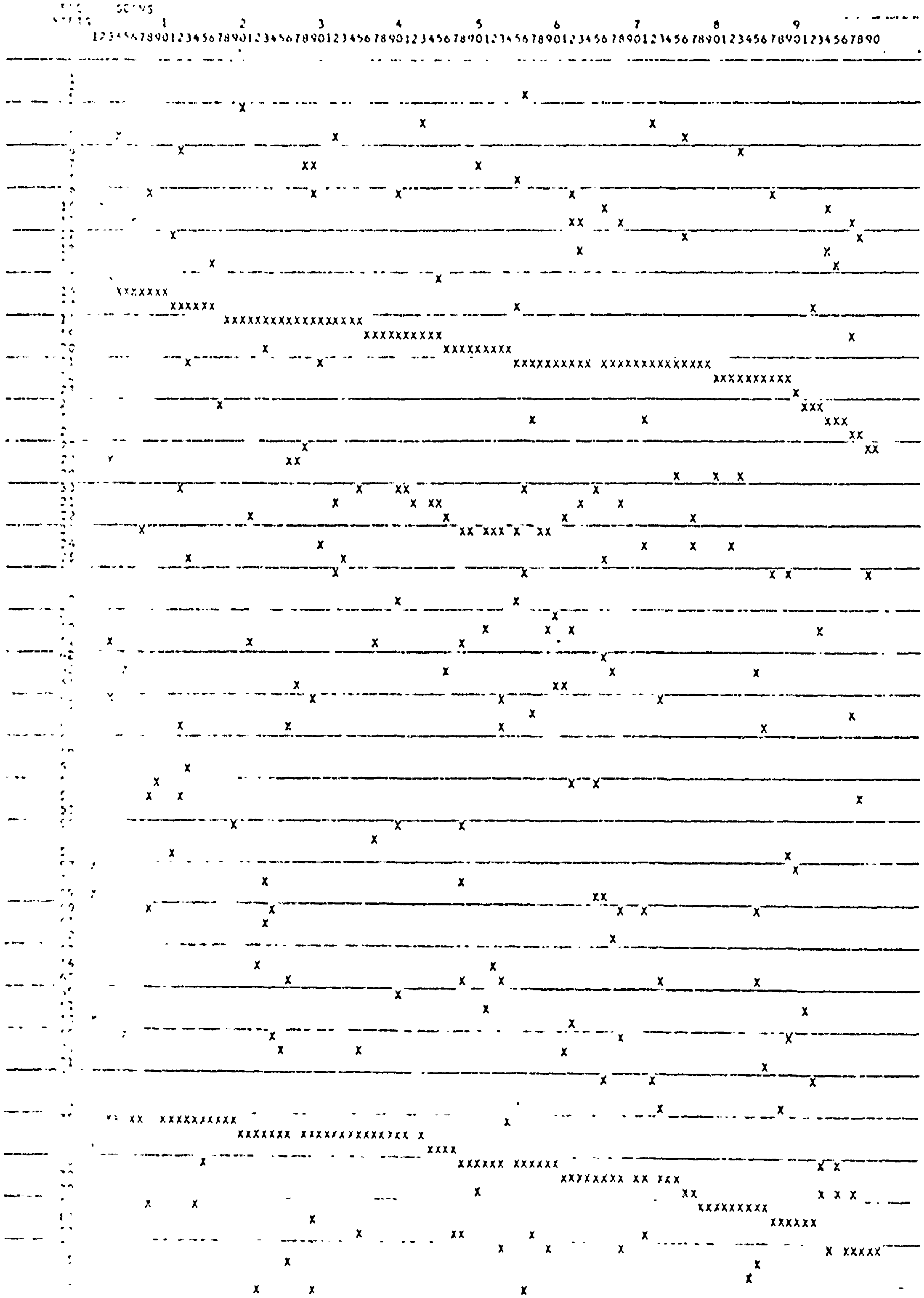


FIGURE 3-2. PRINTOUT OF WIRE-LIKE OBSTACLES AND SINGLE SITES

The array of ones (starting at scan 91) might have been as follows.

```
XX
  X
  XX XXXXX
```

If the new line had continued for one more scan, it also would have qualified as a wire-like obstacle.

- A third wire-like obstacle displays a more common pattern; it has terminated at element 27 on scan 100 after 98 occurrences.

Figure 3-2 shows the picture of the wire-like obstacles and the background single sites.

3.4 OPERATING INSTRUCTIONS

The program is written in standard Fortran IV for IBM 360/370 series computers. There should be no trouble in transferring the main program to other computer systems accepting Fortran. The random number generator is taken from IBM's System 360 Scientific Subroutine Package, Version II. A computer system having a different architecture would require a different random number generator.

4.0 MICROPROCESSOR IMPLEMENTATION

The wire-detection algorithm was implemented on a microprocessor in order to demonstrate the feasibility of such hardware. Since the project was developmental rather than hardware oriented, it was determined that an Intel 8080 MDS (Microprocessor Development System) would be used and that programming would be done in PL/M, Intel's high-level programming language.

4.1 HARDWARE ENVIRONMENT

The components of the MDS system include the following:

- Intel 8080 MDS Processor
- Dual Drive Floppy Diskette
- Keyboard CRT
- Line Printer

The peripheral equipment, similar to that shown in Figures 4-1, 4-2 and 4-3, was specifically selected for use in this developmental system and was not intended to be used in an actual wire-warning system. The MDS components are used as follows:

Intel 8080 MDS Processor

The Intel 8080 MDS Central Processor contains the 8080 microprocessor, interfaces to the various peripherals and 64K bytes of random access memory. All programming for the wire-detection algorithm has been written in PL/M.

Keyboard CRT

This device is the man-machine interface and is the input medium during program development. It is used

MODEL 210
INTELLEC® SERIES II
MICROCOMPUTER DEVELOPMENT SYSTEM

Low-cost development system for MCS-80, MCS-85 and MCS-48 microprocessor families

Compact 4-slot chassis

Single LSI electronics board with CPU, 32K bytes RAM memory and 4K bytes ROM memory

Built-in interfaces for TTY, CRT, Printer, High-Speed Paper Tape Reader/Punch and Universal PROM Programmer

Eight-level nested, maskable priority interrupt system

ROM-based Monitor, Assembler and Editor

Self-Test Diagnostic capability

Standard MULTIBUS™ with multiprocessor and DMA capabilities

Easy upgrade to other Intellec Series II Systems

Compatible with standard Intellec/iSBC Expansion Modules

Software compatible with previous Intellec systems

The Intellec Series II Model 210 Microcomputer Development System is a low-cost, fully-supported development system providing basic hardware and software support for development of products based around Intel's MCS-80 or MCS-85 microprocessor families. Through optional software, this development capability can be extended to products based on the MCS-48 family of microprocessors.

Using the user-supplied system console (TTY or equivalent), the product designer may enter and correct his program's source code, then assemble and begin execution, all using the Model 210 ROM-resident Editor/Assembler. MCS-80 and MCS-85 debugging is accomplished through system monitor debug commands. Completed programs may be punched to paper tape for loading into the user's system or programmed into PROM using the optional Intellec Universal PROM Programmer.

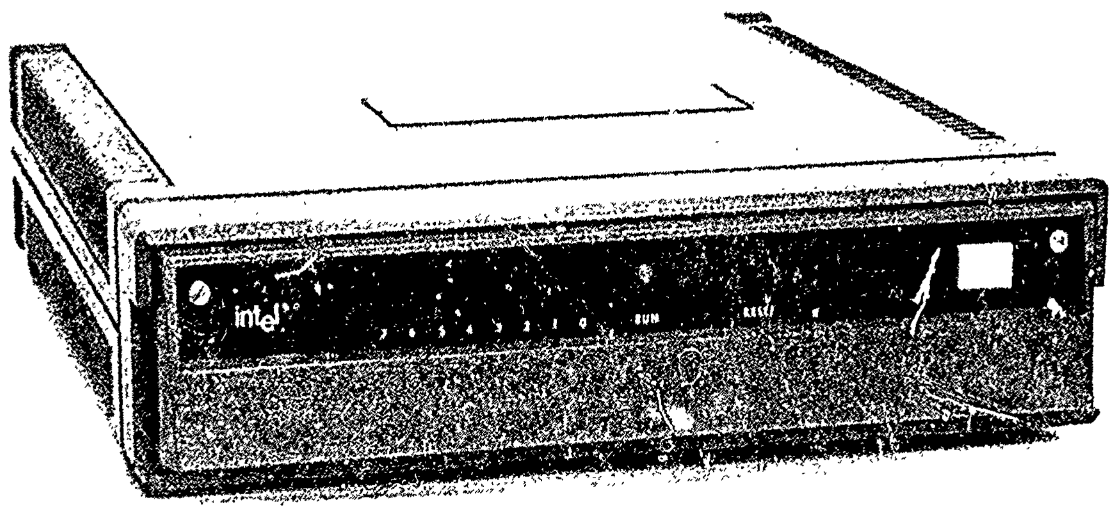


FIGURE 4-1. MDS PROCESSOR

MODEL 220
INTELLEC® SERIES II
MICROCOMPUTER DEVELOPMENT SYSTEM

Complete Microcomputer Development System in one package for MCS-80, MCS-85 and MCS-48 microprocessor families

Integral CRT with detachable upper/lower case "typewriter" style full ASCII keyboard

Integral 250K-byte floppy disk with total storage capacity expandable to over 2M bytes

Single LSI electronics board with CPU, 32K bytes RAM memory and 4K bytes ROM memory

Built-in interfaces for High-Speed Paper Tape Reader/Punch, Printer and Universal PROM Programmer

Eight-level nested, maskable priority interrupt system

Powerful ISIS-II Diskette Operating System with Relocating Macro Assembler, Linker and Locater

Self-Test Diagnostic capability

Standard MULTIBUS with multiprocessor and DMA capability

Compatible with standard Intellec/iSBC Expansion Modules

Software compatible with previous Intellec Systems

The Intellec Series II Model 220 is a complete microcomputer development system integrated into one compact package. It includes a CPU with 32K bytes of RAM memory, 4K bytes of ROM memory, a 2000-character CRT, detachable full ASCII keyboard with cursor controls and upper/lower case capability, and a 250K-byte floppy diskette drive.

Powerful ISIS-II Diskette Operating System software allows the Model 220 to be used quickly and efficiently for assembly and debugging of programs for Intel's MCS-80, MCS-85 or MCS-48 microprocessor families without the need for handling paper tape. ISIS-II performs all file handling operations for the user, leaving him free to concentrate on the details of his own application. When used in conjunction with an optional in-circuit emulator (ICE™) module, the Model 220 provides all the hardware and software development tools necessary for the rapid development of a microcomputer based product.

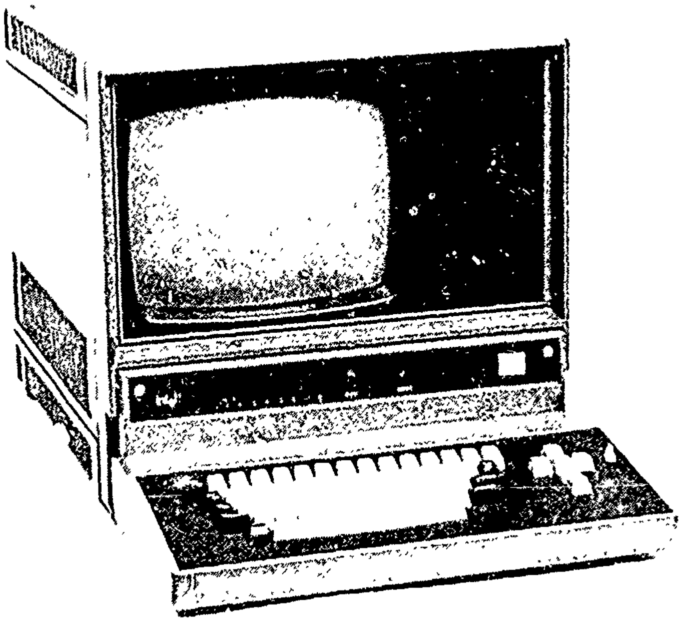


FIGURE 4-2. KEYBOARD CRT

intel®

MODEL 770 PRINTER
INTELLEC® SERIES II
MICROCOMPUTER DEVELOPMENT SYSTEM

Low-cost, hard-copy printer for CRT-based systems

Prints original plus 4 copies

Prints 60 cps (21–90 lines per minute)

5 x 7 dot matrix character format

Tractor feed (rear or bottom feed)

Line width adjustable from 80 to 132 columns on 8½" line

The Model 770 Printer is a low-cost, hard-copy printer designed for use with CRT-based Intellec Series II and Intellec Microcomputer Development Systems. Unidirectional printing at 60 cps makes the Model 770 an ideal printer for the microcomputer-based system designers with small-to-medium printing requirements. The 8½" line width may be filled with 80 to 132 characters by varying the character size.

The printer uses standard fanfold paper through a tractor-feed mechanism to produce an original and up to four copies. Paper can be fed from the bottom or rear of the printer for versatility in any lab environment.

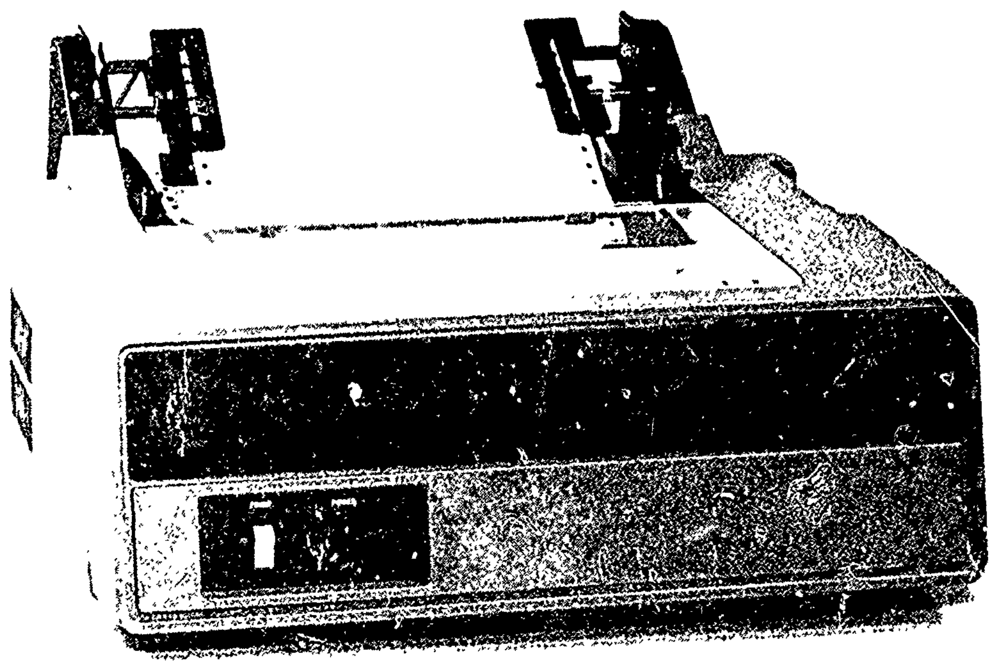


FIGURE 4-3. LINE PRINTER

to control the MDS ISIS-II Operating system and to enter parameters into the wire-detection programs.

Line Printer
This device is the primary output medium during program development. In addition, it is used to print the actual 'picture' of the single sites and possible wires which occur in the background. In the wire-detection program, a condition leading to wire-determination is printed on the Line Printer.

Dual Drive Floppy Diskette
The role of diskettes in the MDS is primarily for storage of the Operating System, Text Editor, PL/M Compiler, and Applications Programs. In the wire-detection application, microprocessor-created backgrounds are also stored on the diskette.

4.2 SOFTWARE IMPLEMENTATION

In the previous section, the contiguous scan algorithm was described as programmed for the IBM 360. In that program, the input sensed by the array was simulated at the beginning of each scan. A different approach was taken for implementation on the micro-processor system. Two programs were developed. The first program creates background scenes with c without wire overlays and stores these scenes on the diskette. The second program can call any particular background scene and will process it using the wire-detection algorithm. The programs are detailed in the following sections.

4.2.1 Background Creation Program

A background creation program was developed to provide scene data for the wire-detection processing. The size of the scene in the microprocessor is limited to 50 array elements and 70 scans. This was done for the simple expediancy of fitting the 'picture' of the scene onto one page of line printer output. This limited size has no effect on the algorithm in the wire-detection program.

Figure 4-4 shows the output of this program. The percentage of background noise (1% in this case) shown in the title, is input to the program via the keyboard by the operator when requested by the CRT display. A random number generator is used in a manner similar to the IBM 360 program. For each background cell, a uniformly distributed random number between 0 and $(2^{16}-1)$ is generated. If the resulting value is less than $(2^{16}-1)$ x input %, a single site is positioned in this cell. Otherwise, there is no single site. (Computer random number generators are more properly referred to as pseudo random number generators and must be written specifically for the architecture of the computer involved. A standard IBM subroutine was used for the IBM 360, but a suitable program for the Intel 8080 was not available). A method described in the literature[4] was used to develop a random number generator for the 8080 in the PL/M language. The program has inserted a space to indicate the absence of a single site and an 'X' to indicate its presence. This printout shows that on scan 1, there is a single site at element 30, on scans 2,3, and 4 there are no single sites, on scan 5, there are sites at elements 8 and 47, etc .

[4] Knuth,D.E., The Art of Computer Programming, Seminumerical Algorithm, Vol.2, Addison-Wesley, Reading, Mass., 1969.

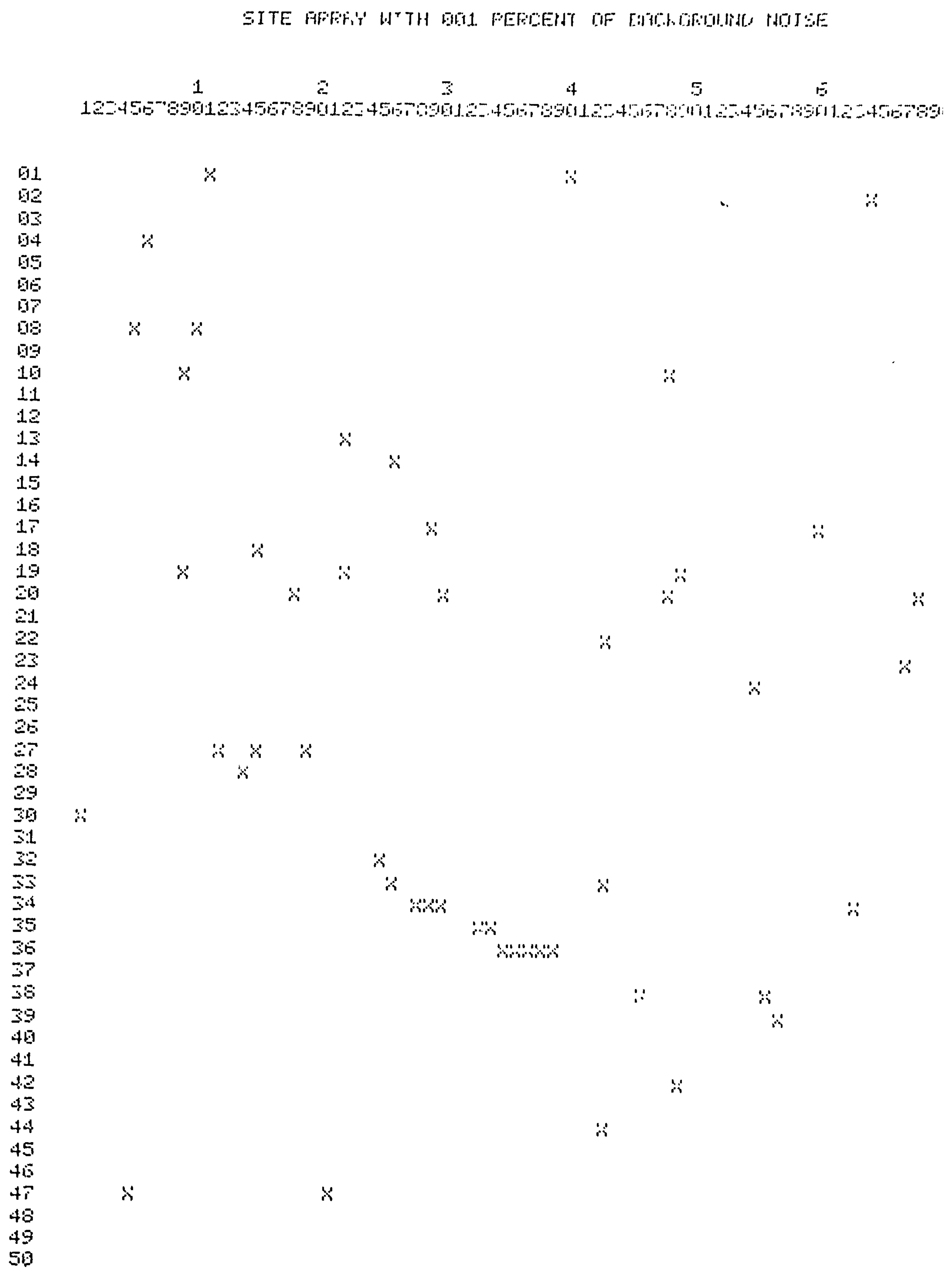


FIGURE 4-4. OUTPUT OF BACKGROUND CREATION PROGRAM

The program then allows the operator to insert additional single sites, which may be positioned to form a wire-like obstacle. In this example, the input wire starts at scan 25, element 32 and terminates at scan 39, element 36. Note that within this line there is one dropout on scan 27 and a 2-scan dropout at scans 31 and 32. The background creation program calls for the operator to give a name to a specific background scene, which may or may not contain a wire-like obstacle. The entire scene is then written to diskette, using standard Intel file management.

4.2.2 Wire Detection Program

The microprocessor wire-detection program can operate on any background scene which has previously been written to diskette. The program initially asks the operator to insert via the CRT keyboard the name of the background file to be examined. The value of the wire-determination criterion is also input via the keyboard on a prompt from the CRT display.

The program reads the background file one scan at a time and searches the scan for single sites. When contiguous sites satisfy the wire-like obstacle definition, a continuity is formed which is tracked until more than 2 scans have been skipped or until the wire-determination criterion is met. In Figure 4-4 the single site which occurs at element 27, scan 12 begins a continuity which drops out on scan 13 and reappears on scans 14 and 15. When the continuity has not appeared in the next 3 scans, (16,17,18) it is dropped. This is very typical of the results obtained in extensive testing of wire-determination criteria in the IBM 360 program. On scan 25, the program picks up a single site which will develop into a wire-like obstacle determiniation. This particular pattern will be defined as a wire-like obstacle for any wire determination

criterion less than 13. The program, however, does not follow the wire to its end (as the 360 program had done), but immediately upon satisfying the criteria, it declares a "WOW ALERT" and prints the location of the last single site in the continuity. The remaining scene is not searched. In Figure 4-5(a), where the input file was the background scene of Figure 4-4 and the wire determination criteria was 6, the alert is given after testing element 35 on scan 33. If no wire determination is made in the entire scene, this fact is printed at the end of the scene processing as in Figure 4-5 (b).

4.3 MICROPROCESSOR OPERATING INSTRUCTIONS

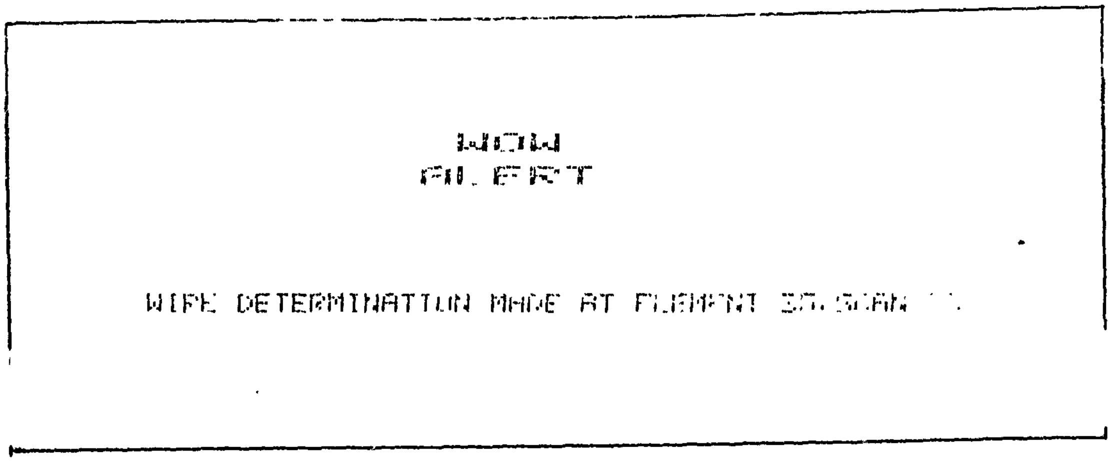
Both of the programs are written in Intel's PL/M and operated under control of Intel's ISIS-II Operating System. Operating System instructions may be found in Intel's instruction manuals. Specific operation of the wire-detection programs follows.

4.3.1 Program "ARRAY"

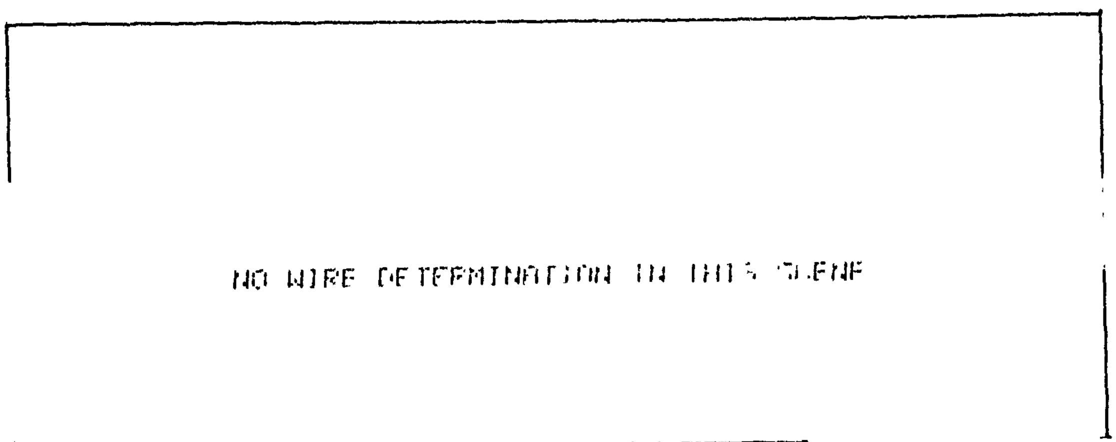
1. In response to the ISIS command prompt (a blinking hyphen on the CRT), the operator keys in the word ARRAY and a carriage return (CR). ISIS will locate the program ARRAY on the diskette, bring it into CPU memory, and transfer control to the program.
2. The program displays

   INPUT A 5 DIGIT NUMBER AND RETURN

   on the CRT. The program is asking the operator for an initializing number for the random number generator. Any value between 00000 and 63777 may be inserted on the keyboard, followed by a CR. (A particular random sequence can always be reproduced by using the same initializing number - hence the term pseudo random



(A) "WOW ALERT" and Wire Location Printout



(B) "NO WIRE" Printout

FIGURE 4-5 EXAMPLES OF WIRE DETECTION
PROGRAM PRINTOUTS

numbers).

3. The program displays

   INPUT A 3 DIGIT NUMBER, REPRESENTING THE PERCENT
   OF BACKGROUND NOISE AND A RETURN

   on the CRT. The operator keys in a value from 000 to 100 followed by a CR. A value of 000 will yield no single sites. A value of 100 would produce the illogical situation of a single site in every position. In practice, values from 000 to 005 have been used. In Figure 4-4, a value of 001 produced 37 random single sites among an array of 3500 elements. (Other initializing values combined with an input of 1% would produce slightly different quantities in different patterns).

4. The system generates the array of 50 rows and 70 columns with the random single sites as described above.

5. The program now allows the operator to input single sites at specific locations in order to form wire-like obstacles. The CRT displays the following message:

   INPUT WIRE
   NOTE: ROW > 70 TERMINATES INPUT WIRE
   INPUT A 2-DIGIT NUMBER FOR ROW LOCATION AND RETURN

   The operator will key in a value from 01 to 50 (followed by CR) to indicate the row of the single site. The CRT will then display:

   INPUT A 2-DIGIT NUMBER FOR COLUMN LOCATION AND
   RETURN

   The operator will key in a value from 01 to 70 (followed by a CR). The program will place a single site in the position determined by the above row and column combination.

6. Step 5 is now repeated as many times as the operator wishes to add additional single sites. When single site input has been completed, the operator responds to the row request with an input value greater than 70, which

informs the program that no more sites are to be processed.

7. The line printer will now print out the 50 x 70 array as in Figure 4-4.
8. The CRT will display

    INPUT FILE NAME AND RETURN

   the operator will key in a file name (according to ISIS standards). The array will be stored on the diskette under the given file name. To give the array the name WIRE1 stored on the diskette 1, the keyboard entry would be

    :F1:WIRE1

   followed by a CR.

4.3.2 Program "Detect"

1. In response to the ISIS prompt (a blinking hypen on the CRT), the operator keys in the word DETECT followed by a CR. ISIS will locate the program DETECT on the diskette, bring it into CPU memory and transfer control to the program.
2. The program will display

    INPUT A 2-DIGIT DETERMINATION CRITERION AND RETURN

   on the CRT. The program is asking for the wire determination criterion. The operator keys in an appropriate value (for example, 07) followed by a carriage return.
3. The program displays

    INPUT FILE NAME AND RETURN

   the operator keys in the file name containing the array to be tested, followed by a CR. The file written as WIRE1 in the previous section would be called by:

    :F1:WIRE1

   followed by a CR.

4. The program reads the file and tests the array to determine whether or not there is a continuous set of single sites which meets the wire determination criterion.
5. If no wire determination is made in the entire array, the line printer will display

   NO WIRE DETERMINATION IN THIS SCENE
6. If a wire determination is made the line printer will display

   WOW ALERT

   WIRE DETERMINATION MADE AT ELEMENT XX, SCAN XX

   The element number corresponds to the row and the scan number corresponds to the column in which the last single site occurred that satisfied the determination criterion.

5.0 NON-CONTIGUOUS SCAN PROGRAM

The previous programs,described in Sections 3 and 4, have dealt with wire-detection procedures in which successive scans were assumed to be contiguous, such that a wire which was detected as a single site at element j or scan n could be expected to be detected on scan n+1 at element j-1, j, or j+1. If there are gaps in the array coverage, however, continuation of a wire from scan n to scan n+1 may fall anywhere within a relatively large window, the size of which depends on the length of the scan gap and the maximum allowable wire angle. Once 2 scans have indicated a possible wire, its position on the next scan can be more accurately predicted. This predicted position must consider not only the previous slope of the possible wire, but also the basic curvature of a catenary type wire-obstacle. The non-contiguous scan algorithm studies were performed with the IBM 360 computer and its implementation is described in the following paragraphs.

5.1 NON-CONTIGUOUS SCAN ALGORITHM

The basic procedures in this non-contiguous scan algorithm program are similar to that of the previous contiguous scan program, i.e., - for each scan; the background noise is determined, the true wire-obstacle sites are overlayed in specific but varying patterns, the elements of the array are searched for single sites, which are then tested as possible continuations of previous sites and/or the beginning of possible new continuities. Any continuity which meets the wire-determination criterion is tracked as long as possible and is printed as output. The details of some of these steps are considerable more complex, however, and other minor program modifications have been made.

The determination of background noise is done by random number generation according to the input percentage value. There is no option for reading the background from cards as in the previous program.

Overlaying of the true wire-obstacle sites involves several steps. The aim in positioning these sites is to approach some realism in approximating the catenary-type path of a wire-obstacle. The program can start overlaying a wire on any particular scan. Associated with each wire input value is a corresponding delta. This determines where the single site for this wire will be located on the next scan. For example, if an input wire single site occurs at element 25 on scan 1, and has a delta value of 5, the corresponding position of the same wire on scan 2 will be 30 and on scan 3, it will be 35. To continue in this manner, however, would produce only straight lines. Therefore, the value of delta must be changed. Modifying the delta value by 1 unit approximately 25% of the time produces a curve that has a hanging wire-like appearance. Dropouts are programmed into the wire overlay procedure at a 5% rate. Both the delta modification and dropout selection are controlled by random number generation, so that various patterns can appear for different wires. The delta procedure causes the slope of a downward wire to become less severe until the wire appears to be horizontal, then the slope will increase as the wire end rises. This may be visualized in Figure 5-2, but it must be understood in this figure that although consecutive scans are printed as contiguous in the scene picture, there are actually substantial gaps between successive scans. Thus to be realistically viewed, the picture should be stretched in the horizontal direction. There is no provision for terminating the wire or turning it around, so all wires eventually run off the 'top' of the figure.

Determination that a single site exists in the scan is a trivial task. Actually, a double site (0110) is acceptable as well as the normal single site (010). Each time a single site is found, 3 procedures must be carried out.

1. - The site location must be entered into a table of possible continuity initializations. That is, the site may be the beginning of a continuity, which fact cannot be determined until future scans. This table is referred to as the 'possibilities' table.
2. - The site may fit into the large window of a single site possibility from the previous scan. For example, a single site is located on scan 5 at element 53 and is entered into the 'possibilities' table. If the gap size is such that the 'big window' is determined to be ±20 pixels, every single site on scan 6, from element 33 to element 73, is a logical continuity of element 53 on scan 5. Thus, if on scan 6, single sites are located at element 58 and 65, the combinations of (53,58) and (53,65) must both be entered into the 'continuity table' as possible wire-like obstacles.
3. - The site may fall within the small window of a previously determined continuity, thus increasing its length (i.e., number of occurrences of that continuity). For example, a single site has been located on scan 5 at element 53, and another on scan 6 at element 58. If this pattern does represent a wire-like obstacle, an entry will be expected on scan 7 at approximately element 63. Because of the changing slope and system error, a window around element 63 is determined, and any single site occurring within that window is accepted as a logical continuation. These window sizes are input to the program; derivations in section 2 yield values of -4 and +2 such that in this example, any single site between elements 59 and 65 would

be acceptable.

Certain ground rules have been established for the process of wire-obstacle prediction. The first 2 entries in a continuity must be successive scans, but after that 1 or 2 dropouts may occur. When a dropout has occurred, the window for the predicted position following the skipped scan is enlarged due to the lengthened gap. Also, when the slope of the wire is close to or equal to zero, the size of the window is cut down in order to diminish the effects of random noise.

Additional problems occur when one of the background false alarm single sites falls within the expected window of a possible wire-obstacle. Two situations can be recognized.

A. - The expected continuity of the wire and a random false alarm both fall within the predictive window. Both paths must be tracked when this occurs. In the previous example in which the window included elements 59 through 65, if a false alarm occurred at element 60 and the actual wire site were at 63, this would require temporarily tracking both paths. The false alarm path will die out after subsequent scans contain no single sites in its predictive window.

B. - The expected continuity of a wire becomes a temporary dropout on the same scan that a false alarm is picked up within the window. Several alternatives are available here. In the current program it has been assumed that both the false alarm rate and the wire dropout rate are sufficiently low, so that their joint probability might be ignored. Any wire terminated in this manner would automatically be re-initiated on the next scan. If the joint probability is considered high enough to be of concern, the program could be modified to track this

situation also.

Thus the program involves a considerable quantity of book-keeping. Each single site must be considered as a possible initial entry of a continuity. Every continuity must be tracked through dropouts and with multiple entries within its predictive window. The size of the predictive window must be enlarged when dropouts occur and tightened when the slope of the wire is close to zero. Any possibility is dropped if there are no entries within its large angular window on the next scan. Continuities are dropped when no entry has been added for 3 successive scans. The criterion for determining that a continuity is indeed a wire-like obstacle is an input value and any continuity which exceeds this value is printed. Once again, however, such continuities are tracked through all scans and the printout occurs only when the continuity is terminated or the last scan has been processed.

5.2 PROGRAM INPUT

Basic input parameters include the number of elements in the array, the number of scans to be processed, the percentage of random background noise and the wire-determination criterion. The window sizes are also input values; a large window determines the first possible leg of a wire-like continutiy, a small window is used for the linear extrapolation for all subsequent legs. The initial locations are input in the following manner; an element number which will become the first single site of the wire-like continuity, a delta value to determine the location of the single site on the next scan, and the scan on which the wire will begin. It is also possible to input a specific series of single sites which may or may not define a wire-obstacle.

Specific input cards are as follows: (All input except the run identification and PBACK are integers, right justified within a 5 column field).

1. 1 card (format 20A4) containing the

   RUN ID — Any alphanumeric data in columns 1-80.

2. 1 card (format 7I5,F10.0) containing the following run parameters:

Field	Name	Data
1-5	IX	Random number initializer,- any odd integer.
6-10	NARRAY	Number of elements in array, NARRAY ≤1728
11-15	NTRUE	Number of true sites to be input, NTRUE ≤10.
16-20	NSCAN	Number of scans, NSCANS ≤ 100
21-25	NCNTPR	Wire determination criteria for printing; i.e., a continuity will be printed if the number of occurrences > NCNTPR.
26-30	NARRPR	Single site map option. If NARRPR = 0, the entire array of elements for all scans is printed. If NARRPR ≠ 0, no printout occurs.
31-35	NDRAW	Extra line option. If NDRAW >0, 2 sets of NDRAW cards with scan values and

Field	Name	Data
		corresponding element values with the read (see item 5). Maximum value, NDRAW = 10. If NDRAW = 0, no additional input sites are read.
36-45	PBACK	The percentage of background sites to be generated by random numbers. PBACK is a fixed point value with decimal point in a 10 column field.

3. 1 card (format 4I5) containing 'window sizes' for the predictive logic algorithm.

Field	Name	Data
1-5 6-10	IWBG1 IWBG2	The size of the predictive window in the assumption of a continuity from the 1st occurrence of a single site to the 2nd occurrence. Both values are input as positive numbers. IWBG1 represents the 'up' direction IWBG2 is 'down'.
11-15 16-20	IWSM1 IWSM2	The size of the smaller predictive window in the assumption of another occurrence of a continuity from scans n-1 and n to scan n+1. Both values are input as positive values determined from a linearly extrapolated position. IWSM1 is the 'up' direction

		and is normally greater than IWSM2, the down direction.

4. NTRUE cards (each with a format 3I5). Each card contains the input data for the initial occurrence of a wire-like obstacle.

1-5	ITRUE	Element position of single site representing the 1st occurrence of wire-like obstacle.
6-10	ITDEL	Incremental value to calculate subsequent position of single sites of wire-like obstacle.
11-15	IFIRST	First scan on which the wire-like obstacle is to occur.

5. For NDRAW >0 only. NDRAW cards (format 16I5) containing the extra scan values, followed by an additional NDRAW cards (format 16I5) containing the corresponding extra element values. Scan and element values are combined to form additional single sites.

5.3 PROGRAM OUTPUT

Figures 5-1 and 5-2 show the printout of a particular set of data processed in the non-contiguous scan program. Figure 5-1 lists the various input values: 100 elements in the array, 100 scans, 2 input sites, 0.4% random background, a wire-determination criteria of 9, and a random number initializer of 11111. Window sizes have been chosen consistent with the derivations of section 2; the initial 'large' angular window is ±16 elements, the small predictive window is +4, -2 (+ is used here as the 'up'

```
0.4 PERCENT BACKGROUND

  100 = LENGTH OF ARRAY
  100 = NUMBER OF SCANS
    2 = SITES TO BE INPUT
  0.4 = PERCENTAGE RANDCM BACKGROUND
    9 = CONTINUITY CRITERIA
11111 = RANDOM NUMBER INITIALIZER


LARGE AND SMALL WINDOW SIZES

  16   16    4    2

ORIGINAL TRUE SITES, DELTAS, AND FIRST SCAN

  25   55
   5    3
   1   40


EXTRA INPUT SITES

 SCAN       72  73  74  75  76  77  79  80  81  82  83  85  86  87  89  90
 ELEMENT    58  61  64  67  69  71  75  77  78  79  80  80  79  77  73  70

            CONTINUITIES
 LAST 2        LAST       TCTAL
ELEMENTS       SCAN     OCCLRANCES

 50  43         31         29
  8   3         42         40
  8   3         61         22
 73  70         90         16
```

FIGURE 5-1. PROGRAM INPUT VALUES FOR THE NON-CONTIGUOUS SCAN ALGORITHM

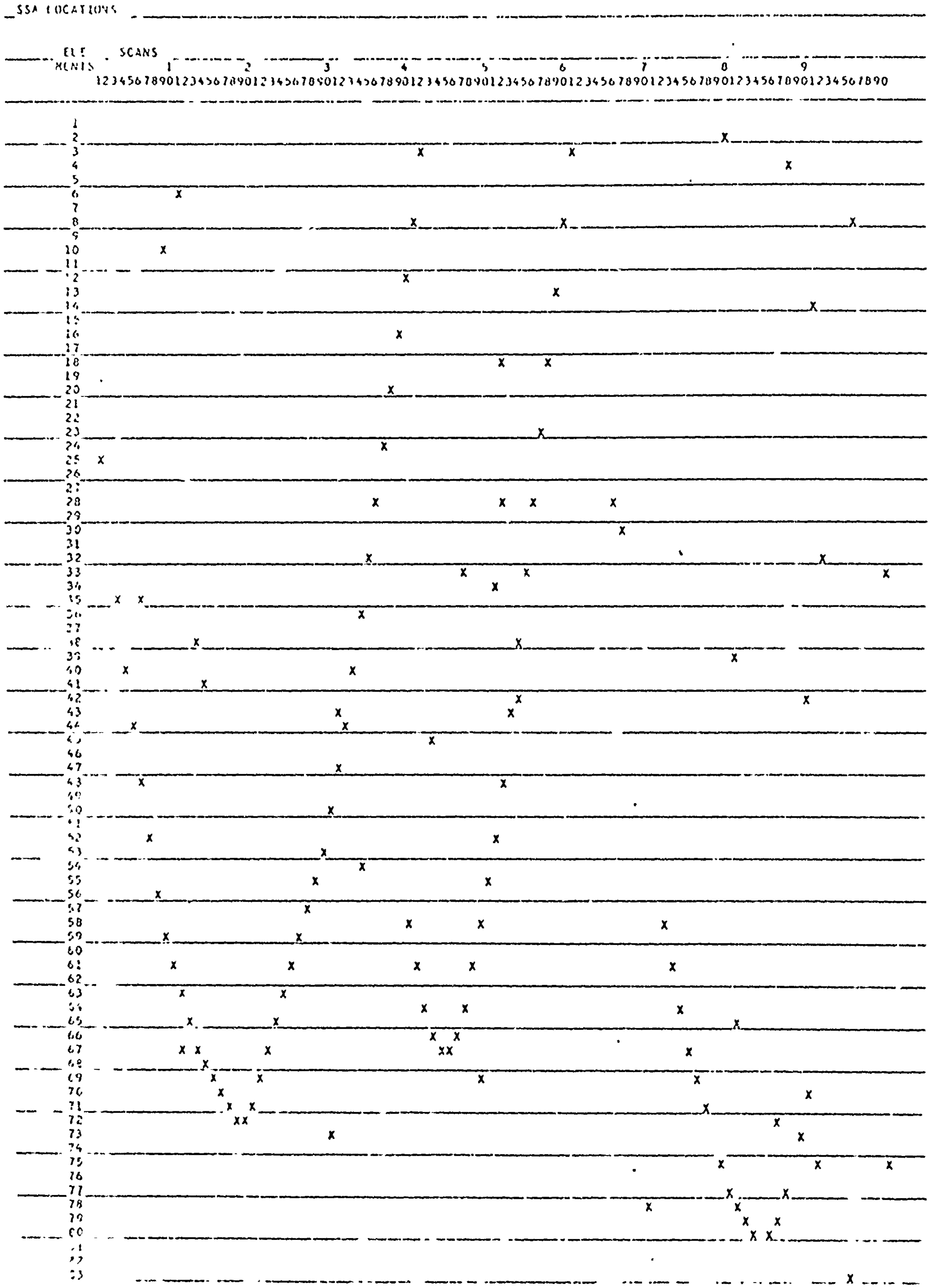


FIGURE 5-2. SAMPLE PROGRAM OUTPUT FROM THE NON-CONTIGUOUS SCAN ALGORITHIM

direction in the 'picture'; in the program, the 'up' direction involves lowering the element numbers).

Two wire initializing sites have been input. On scan 1, a single site will appear at element 25 and with a delta value of 5, it should thus appear at element 30 on scan 2. Note in Figure 5-2 that this site does not appear; it has been one of the 5% which are randomly dropped. Because 2 consecutive scans are required to initiate a continuity, the program will discard the site on scan 1 assuming it to be a false alarm. The wire-obstacle overlay picks up again at element 35 on scan 3, element 40 on scan 4, and then with a change of delta, shows as elements 44, 48, 52, etc., on successive scans. The second input wire is scheduled to start at element 55 on scan 40. Actually, it starts at element 58 on the 40th scan because after the first scan, the program adds the current delta value (3) to the previous element value (55) to obtain the new element (58) to be placed at the current scan.

Figure 5-2 shows that both of these input wires follow a randomly controlled catenary-like path. Since there is no termination or reversal, the wires disappear off the 'picture' at the top of the scan. An additional wire-like obstacle has also been input to the program using the scan/element input option. It starts at element 58, scan 72 and ends at element 70 on scan 90. Note that there are dropouts on scans 78,84,and 88. This curve can be followed precisely in Figure 5-2.

The wire continuity printout has occurred for every path in which there are 9 or more occurrences. Referring to the input values and an inspection of Figure 5-2, there should be 3 wire-obstacle determinations in this run. However, the program printout shows 4 such continuities.

- The first listing is caused by the condition explained in section 5.1 in which both a random false alarm and the true wire site occurred within the predictive window. The entry shows that the continuity terminated at scan 31, with 29 occurrences and that the last 2 element numbers were 50 and 43, (43 being the terminating element). Reference to Figure 5-2 shows that the single site at element 43 on scan 31 was a false alarm. By reviewing the algorithm logic, we can see why it was picked up. Tracking this wire previously, we had

scan	29	30
element	53	50

  The next expected linear value would be element 47 on scan 31. However, the window is from 43 to 49. Thus, the false alarm at element 43 is picked up as a logical value. On the next scan, however, the expected linear continuation from 50 to 43 would be at element 36 with a window of 32 to 38. Since no single site is picked up on this or the following scan, the path terminates. If this false alarm pickup had terminated prior to meeting the continuity criteria, it would not have been printed. However, 29 occurrences had already been counted, so a wire-obstacle determination was made.
- The next entry in the continuity printout table shows the 'true' path of the same wire, continuing until it disappears from the 'picture' at scan 42.
- The third entry shows the second input wire, which also runs off the 'picture'.
- The last entry shows the tracking of the additional wire-like obstacle in which each single site was individually input. This continuity has been tracked through several dropouts.

The output of this non-contiguous scan program has innumerable patterns depending on the wire-obstacle overlays and the random number generation. Intersecting wires can also be traced and for proper visualization, the picture should be stretched in a horizontal direction. Increasing the percentage of false alarms greatly increases the quantity of the bookkeeping and false paths, but all actual wire-like obstacles are tracked.

5.4 OPERATING INSTRUCTIONS

The program is written in standard Fortran IV for IBM 360/370 series computers. The random number generator is taken from IBM's System 360 Scientific Subroutine Package, Version Il. A computer system having a different architecture would require a different random number generator.

6.0 SUMMARY AND CONCLUSIONS

The studies and experiments performed to define a logic and display system, compatible with Single Site Activation signals from a wire detection system, have established a viable algorithm for the detection of wire-like objects under realistic scene conditions. The key results and the conclusions derived are summarized as follows:

- In the contiguous scan computer program it was shown that two input "wires" were successfully tracked under simulated conditions, i.e., 2.1% random background, one pixel movement of wire 10% of the frame time, "double" hits 10% of the time and dropouts 5% of the time. These results showed that the algorithm was not foiled by these disturbance conditions.

- The hardware implementation of the contiguous scan mode of operation provided a powerful test system for verifying the results of the computer simulations. This equipment was delivered to USAECOM, Ft. Monmouth as part of this contract. A major capability of the test system, i.e., the background creation program permits analysis of various wire inputs against stored random background programs and displays the processed results.

- The non-contiguous computer program proved out, primarily, the "window" concept and wire continuity criteria for the recognition of wires under realistic conditions, i.e., large spaces between successive scans and catenary-type wire obstacles.

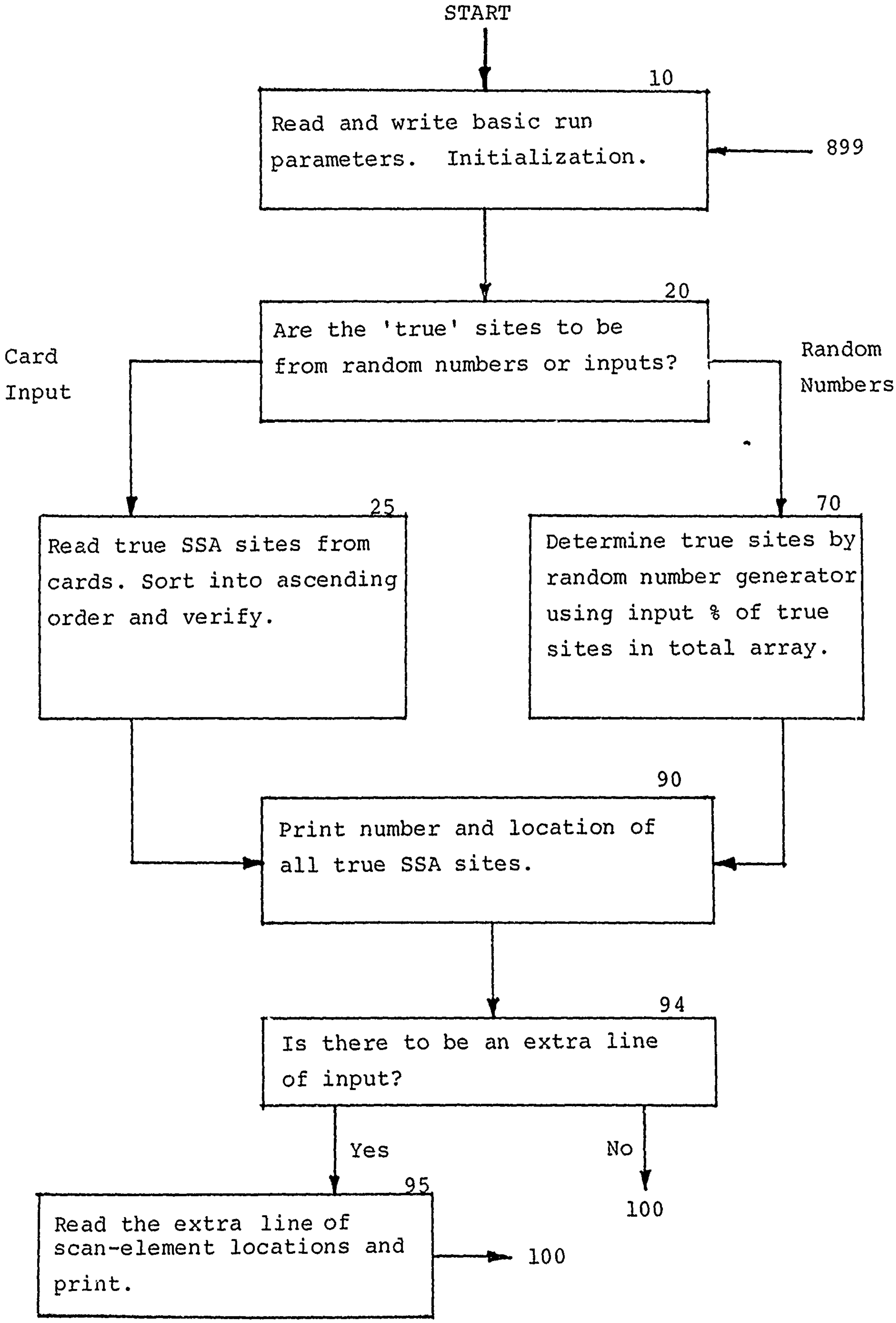
- The work performed in this study indicates that a wire pattern recognition system using the developed algorithms is a realizable concept. The algorithms are scalable upwards

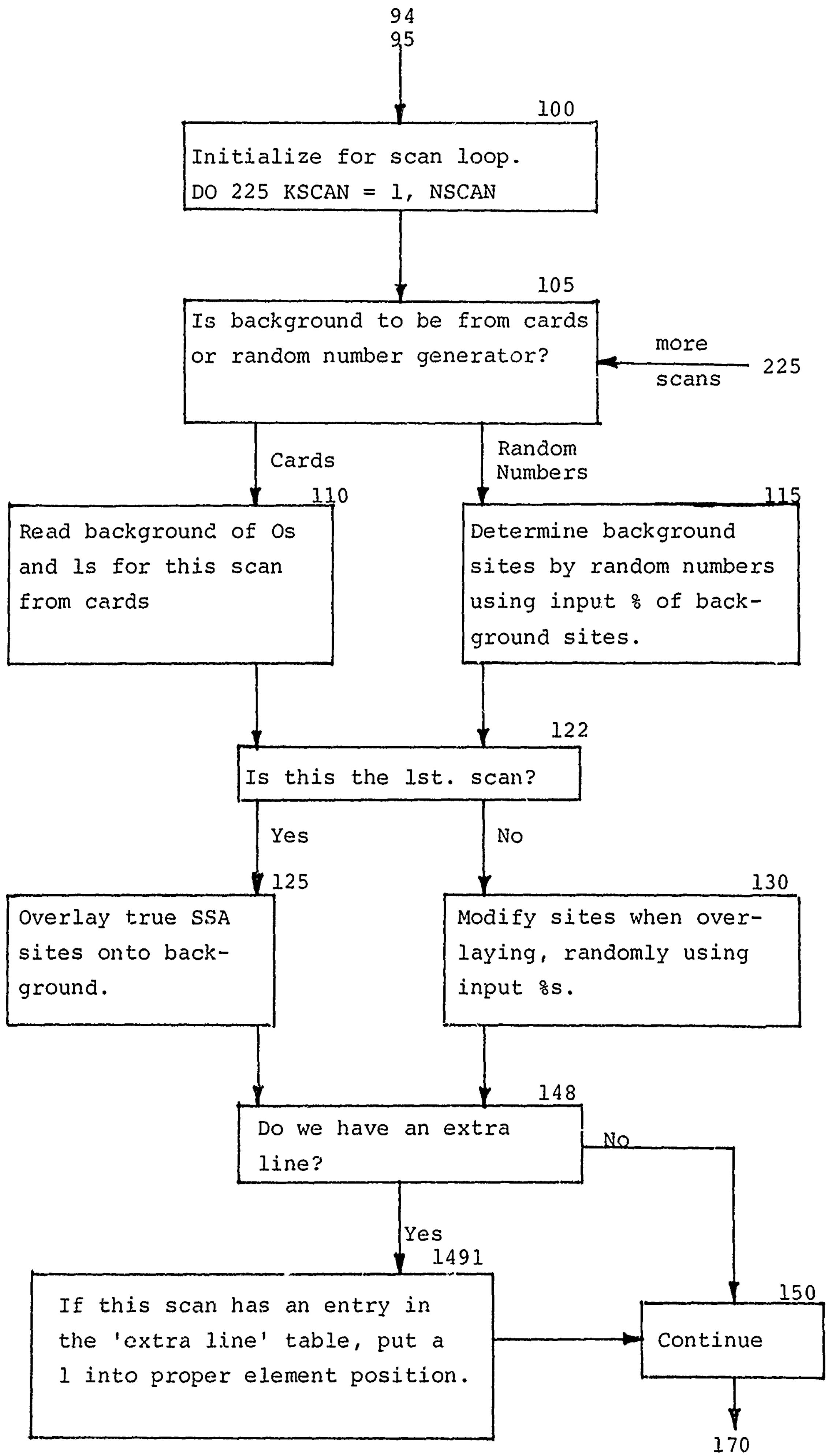
in terms of memory capacity and higher speed processors are currently avaialble to make the system operate in "real time". It is concluded that the combination of a CCD wire detection technique, described in the earlier referenced wire object detection study,[5] and the described wire pattern recognition algorithms would provide the major components of a practical wire object detection/ recognition warning system.

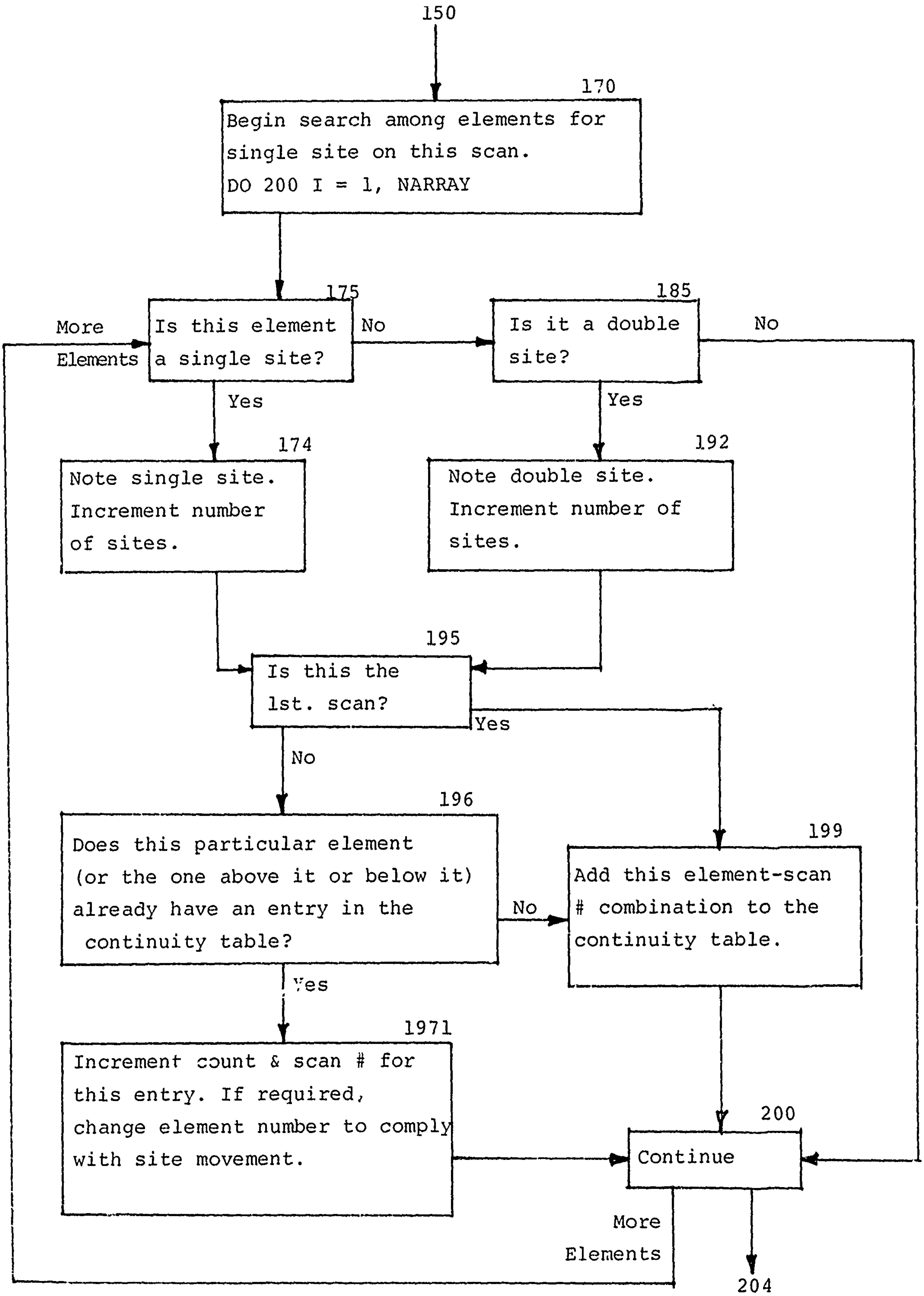
[5] Kleehammer, op.cit.

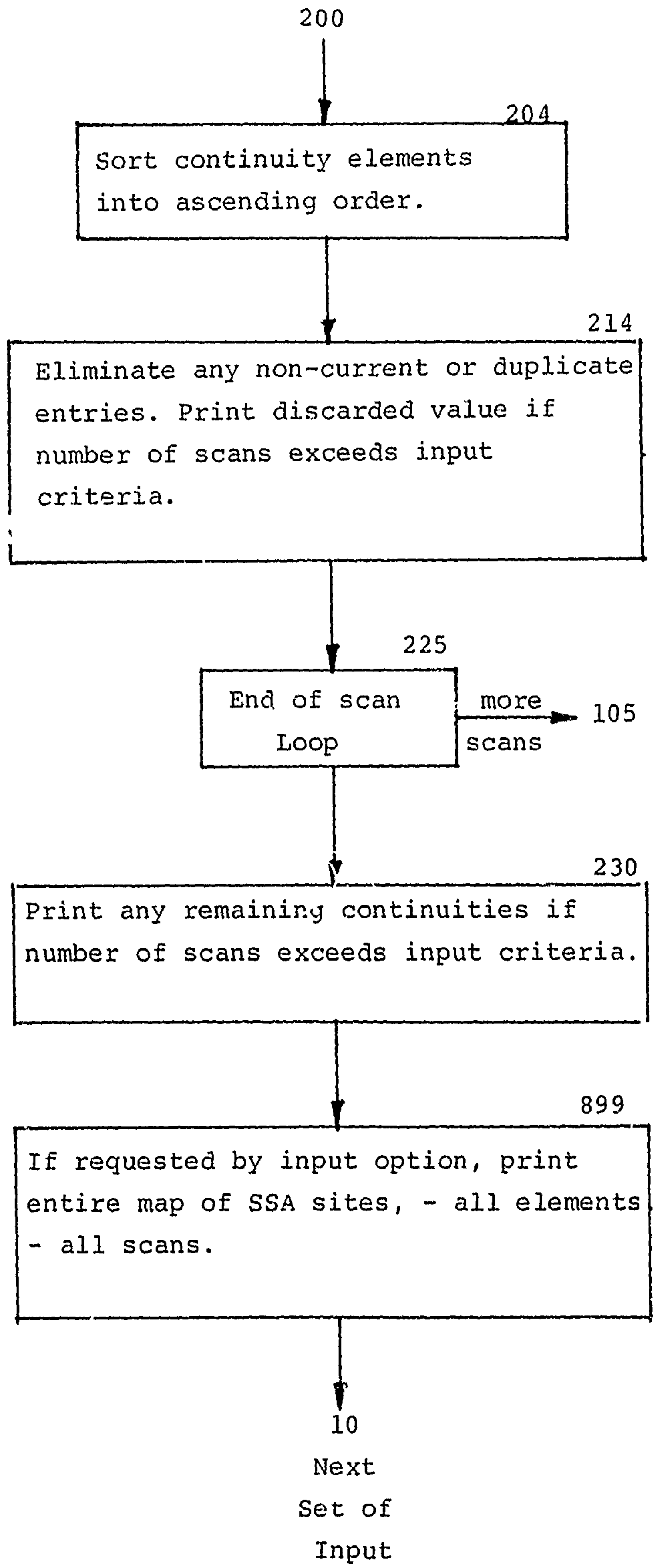
APPENDIX A

CONTIGUOUS SCAN PROGRAM


START
10
Read and write basic run parameters. Initialization.
899
20
Are the 'true' sites to be from random numbers or inputs?
Card Input
Random Numbers
25
Read true SSA sites from cards. Sort into ascending order and verify.
70
Determine true sites by random number generator using input % of true sites in total array.
90
Print number and location of all true SSA sites.
94
Is there to be an extra line of input?
Yes
No
100
95
Read the extra line of scan-element locations and print.
100


94
95
100
Initialize for scan loop.
DO 225 KSCAN = 1, NSCAN
105
Is background to be from cards or random number generator?
more scans
225
Cards
Random Numbers
110
Read background of 0s and 1s for this scan from cards
115
Determine background sites by random numbers using input % of background sites.
122
Is this the 1st. scan?
Yes
No
125
Overlay true SSA sites onto background.
130
Modify sites when overlaying, randomly using input %s.
148
Do we have an extra line?
No
Yes
1491
If this scan has an entry in the 'extra line' table, put a 1 into proper element position.
150
Continue
170


150
170
Begin search among elements for single site on this scan. DO 200 I = 1, NARRAY
175
Is this element a single site?
More Elements
No
185
Is it a double site?
No
Yes
174
Note single site. Increment number of sites.
Yes
192
Note double site. Increment number of sites.
195
Is this the 1st. scan?
Yes
No
196
Does this particular element (or the one above it or below it) already have an entry in the continuity table?
No
199
Add this element-scan # combination to the continuity table.
Yes
1971
Increment count & scan # for this entry. If required, change element number to comply with site movement.
200
Continue
More Elements
204


200
204
Sort continuity elements into ascending order.
214
Eliminate any non-current or duplicate entries. Print discarded value if number of scans exceeds input criteria.
225
End of scan Loop
more scans
105
230
Print any remaining continuities if number of scans exceeds input criteria.
899
If requested by input option, print entire map of SSA sites, - all elements - all scans.
10
Next Set of Input

```
C     SINGLE SITE ACTIVATION     MOVING LINES
C
      DIMENSION ITRUE(50), IARRAY(1729), JSSA(200,100),RUNID(20)
      DIMENSION PLINE(100), NSSA(100),NPLACE(100), JCONT(3,1000)
      DIMENSION ISCAN(16),IELEM(16)
      DATA SPACE/' '/,STAR/'X'/
C
C     READ INPUT PARAMETERS
   10 READ (1,11,END=999) RUNID
   11 FORMAT (20A4)
      READ (1,12) IX,NARRAY,NTRUE,NSCAN,NCNTPR,NARRPR,NDRAW,PBACK,PTRUE
   12 FORMAT (7I5,2F10.0)
C     READ PERCENTAGES FOR WAVYNESS
   13 READ (1,14) PSAME,PUP,PDOWN,PPUP,PPDOWN, PDROP
   14 FORMAT (6E10.0)
      WRITE (3,15) RUNID,NARRAY,NSCAN,PSAME,NTRUE, PUP,PTRUE,PDOWN,
     1 PBACK,PPUP,NCNTPR,PPDOWN,IX,PDROP
   15 FORMAT (1H1///10X,20A4///10X,I6,18H = LENGTH OF ARRAY,22X,
     132HPERCENTAGE OF TRUE SITE MOVEMENT//
     210X,I6,18H = NUMBER OF SCANS,30X,F6.1,13H  NO MOVEMENT//
     310X,I6,20H = SITES TO BE INPUT,28X,F6.1,11H  MOVE UP 1//
     410X,F6.1,31H = PERCENTAGE RANDOM TRUE SITES,17X,F6.1,13H  MOVE DOW
     5N 1//
     610X,F6.1,31H = PERCENTAGE RANDOM BACKGROUND,17X,F6.1,11H  DOUBLE-U
     7P//
     810X,I6,22H = CONTINUITY CRITERIA,26X,F6.1,13H  DOUBLE-DOWN//
     910X,I6,28H = RANDOM NUMBER INITIALIZER,20X,F6.1,10H  DROP-OUT//)
   16 DO 17 I = 1,200
      DO 17 K = 1,100
   17 JSSA(I,K) = 0
      IARRAY(NARRAY+1) = 0
      NAR1 = NARRAY - 1
C     MAKE PERCENTAGES CUMULATIVE
   18 PUP = PUP + PSAME
      PDOWN = PDOWN + PUP
      PPUP = PPUP + PDOWN
      PPDOWN = PPDOWN + PPUP
C
C     CHECK FOR TYPE OF SSA INPUT
   20 IF (NTRUE) 70,70,25
C     READ SSA SITES FROM CARDS, THEN SORT AND CHECK
   25 READ (1,26) (ITRUE(J),J=1,NTRUE)
   26 FORMAT (16I5)
      IF (NTRUE-1) 90,90,28
   28 NTRUE1 = NTRUE - 1
   30 DO 36 I = 1,NTRUE1
      I1 = I + 1
   32 DO 36 J = I1,NTRUE
      IF (ITRUE(I)-ITRUE(J))  36,36,34
   34 TEMP = ITRUE(I)
      ITRUE(I) = ITRUE(J)
      ITRUE(J) = TEMP
   36 CONTINUE
C     TEST FOR ILLEGAL VALUES
```

```
0035            40 IF (ITRUE(1)-1) 42,42,46
0036            42 DO 44 J = 2,NTRUE
0037            44 ITRUE(J-1) = ITRUE(J)
0038               NTRUE = NTRUE-1
0039               GO TO 50
0040            46 IF (ITRUE(NTRUE)-NARRAY) 50,48,48
0041            48 NTRUE = NTRUE-1
0042            50 DO 52 I = 2,NTRUE
0043               IF (ITRUE(I)-ITRUE(I-1)-1) 56,56,52
0044            52 CONTINUE
0045               GO TO 60
0046            56 NTRUE = NTRUE-1
0047               DO 58 J = I,NTRUE
0048            58 ITRUE(J) = ITRUE(J+1)
0049               GO TO 50
0050            60 GO TO 90
              C    SSA SITES FROM RANDOM NUMBERS
0051            70 CONTINUE
0052               NTRUE = 0
0053               I = 2
0054            74 CALL RANDU(IX,IY,YFL)
0055               IX = IY
0056               YFL = YFL*100.
0057               IF (YFL-PTRUE) 76,76,78
0058            76 NTRUE = NTRUE+1
0059               ITRUE(NTRUE) = I
0060               I = I+1
0061            78 I = I+1
0062               IF (I-NARRAY) 74,80,80
0063            80 CONTINUE
              C
              C    PRINT SSA LOCATIONS
0064            90 WRITE (3,91) NTRUE, (ITRUE(J),J = 1,NTRUE)
0065            91 FORMAT (//10X,42HCOMPUTED OR INPUT SSA LOCATIONS    TOTAL =,I4//
                  1(10X,20I5/))
              C    CHECK FOR INPUT OF EXTRA LINE
0066            94 IF (NDRAW) 99,99,95
0067            95 READ (1,96) (ISCAN(J),J=1,16)
0068            96 FORMAT (16I5)
0069               READ (1,96) (IELEM(J),J=1,16)
0070            97 WRITE (3,98) (ISCAN(J),J=1,16),(IELEM(J),J=1,16)
0071            98 FORMAT (///10X,16HEXTRA INPUT LINE//12X,8HSCAN    ,16I6//
                  112X,8HELEMENT ,16I6//)
0072            99 CONTINUE
0073               WRITE (3,92)
0074            92 FORMAT (///17X,12HCONTINUITIES//10X,26HELEMENT      TOTAL    LAST/
                  111X,25HNUMBER   OCCURANCES  SCAN//)
              C
              C    SET UP SCAN LOOPS
0075           100 NCONT = 0
0076               DO 225 KSCAN = 1,NSCAN
              C    IS BACKGROUND FROM CARDS OR RANDU
0077           105 IF (PBACK) 110,110,115
              C    BACKGROUND FROM CARDS
```

```
0078          110 READ (1,111) (IARRAY(I),I=1,NARRAY)
0079          111 FORMAT (80I1)
0080              GO TO 122
              C   BACKGROUND FROM RANDOM NUMBERS
0081          115 CONTINUE
0082          116 DO 120 I=1,NARRAY
0083              CALL RANDU(IX,IY,YFL)
0084              IX = IY
0085              IARRAY(I) = 0
0086              YFL = YFL*100.
0087              IF (YFL-PBACK) 116,118,120
0088          118 IARRAY(I) = 1
0089          120 CONTINUE
              C
0090          122 IF (KSCAN-1) 125,125,130
              C   OVERLAY TRUE SSA LOCATIONS ONTO BACKGROUND
0091          125 DO 128 K = 1,NTRUE
0092              J = ITRUE(K)
0093          128 IARRAY(J) = 1
0094              GO TO 148
              C
              C   AFTER FIRST SCAN, MODIFY SITES WHEN OVERLAYING
0095          130 DO 146 K = 1,NTRUE
0096              J = ITRUE(K)
0097              IF (J-2) 138,138,131
0098          131 IF (J-NAR1) 132,138,138
0099          132 CALL RANDU(IX,IY,YFL)
0100              IX = IY
0101              YFL = YFL*100.
0102          133 IF (YFL-PSAME) 138,138,134
0103          134 IF (YFL-PUP) 135,135,136
0104          135 J = J + 1
0105              GO TO 138
0106          136 IF (YFL-PDOWN) 137,137,140
0107          137 J = J-1
0108          138 IARRAY(J) = 1
0109              ITRUE(K) = J
0110              GO TO 146
0111          140 IF (YFL-PPUP) 144,144,141
0112          141 IF (YFL-PPDOWN) 143,143,142
0113          142 IARRAY(J) = 0
0114              GO TO 146
0115          143 J = J-1
0116          144 IARRAY(J) = 1
0117              IARRAY(J+1) = 1
0118          146 CONTINUE
0119          148 CONTINUE
              C
              C   CHECK FOR EXTRA LINE
0120          149 IF (NDRAW) 150,150,1491
0121         1491 DO 1492 J = 1,16
0122              IF (KSCAN-ISCAN(J)) 1492,1493,1492
0123         1492 CONTINUE
0124              GO TO 150
```

```
0125      1493 JJ = IELEM(J)
0126           IARRAY(JJ) = 1
0127       150 CONTINUE
        C
        C      BEGIN SEARCH FOR SINGLE SITES
0128       170 NSITES = 0
0129       172 DO 200 I = 2,NARI
0130           IF (IARRAY(I)-1) 200,174,200
        C      CHECK FOR SINGLE
0131       174 IF (IARRAY(I-1)+IARRAY(I+1)) 185,175,185
0132       175 NSITES = NSITES + 1
0133           JSD = 1
0134           GO TO 195
        C      CHECK FOR DOUBLE
0135       185 IF (IARRAY(I)+IARRAY(I+1)-2) 200,186,200
0136       186 IF (IARRAY(I-1)+IARRAY(I+2)) 200,192,200
        C      HAVE DOUBLE
0137       192 NSITES = NSITES + 1
0138           JSD = 2
        C      CONTINUITY BOOKKEEPING
0139       195 IF (KSCAN-1) 199,199,196
0140       196 DO 197 KCONT = 1,NCONT
0141      1961 IF (JCONT(1,KCONT)-I) 1962,1964,1962
0142      1962 IF (JCONT(1,KCONT)-(I-1)) 1963,1964,1963
0143      1963 IF (JCONT(1,KCONT) -(I+1)) 197,1971,197
0144      1964 IF (JCONT(1,KCONT+1)-(I+1)) 1965,1966,1965
0145      1965 IF (JCONT(1,KCONT+1)-I) 1981,1966,1981
0146      1966 IF (JCONT(2,KCONT)-JCONT(2,KCONT+1)) 197,197,1981
0147       197 CONTINUE
0148           GO TO 199
0149      1971 IF (JSD-1) 198,1981,198
0150       198 JSSA(NSITES,KSCAN) = I + 1
0151           GO TO 1982
0152      1981 JCONT(1,KCONT) = I
0153           JSSA(NSITES,KSCAN) = I
0154      1982 JCONT(2,KCONT) = JCONT(2,KCONT)+1
0155           JCONT(3,KCONT) = KSCAN
0156           GO TO 200
0157       199 NCONT = NCONT+1
0158           JCONT(1,NCONT) = I
0159           JCONT(2,NCONT) = 1
0160           JCONT(3,NCONT) = KSCAN
0161           JSSA(NSITES,KSCAN) = I
        C      END OF I ELEMENT LOOP
0162       200 CONTINUE
0163       202 NSSA(KSCAN) = NSITES
        C
        C      FINISH SCAN
        C      SORT CONTINUITIES
0164       204 IF (NCONT-1) 225,225,205
0165       205 NCONT1 = NCONT-1
0166       206 DO 212 IJ = 1,NCONT1
0167           IJ1 = IJ + 1
0168       208 DO 212 IK = IJ1,NCONT
```

```
0169                IF (JCONT(1,IJ)-JCONT(1,IK)) 212,212,210
0170            210 ITEMP1 = JCONT(1,IJ)
0171                ITEMP2 = JCONT(2,IJ)
0172                ITEMP3 = JCONT(3,IJ)
0173                JCONT(1,IJ) = JCCNT(1,IK)
0174                JCONT(2,IJ) = JCONT(2,IK)
0175                JCONT(3,IJ) = JCONT(3,IK)
0176                JCONT(1,IK) = ITEMP1
0177                JCONT(2,IK) = ITEMP2
0178                JCONT(3,IK) = ITEMP3
0179            212 CONTINUE
              C     ELIMINATE ANY NON-CURRENT CR DUPLICATE ENTRIES
0180            214 I = 1
0181            215 IF (KSCAN-JCONT(3,I)-2) 2151,2151,216
0182           2151 IF (I-NCONT) 2152,220,220
0183           2152 IF (JCONT(1,I)-JCONT(1,I+1)) 2153,2154,2154
0184           2153 I = I+1
0185                GO TO 215
0186           2154 JCONT(2,I) = MAXO(JCONT(2,I),JCONT(2,I+1))
0187                JCONT(3,I) = MAXO(JCONT(3,I),JCONT(3,I+1))
0188                I = I+1
0189                GO TC 2170
              C     PRINT DISCARDS IF MORE THAN CERTAIN NUMBER OF SCANS
0190            216 IF (JCONT(2,I)-NCNTPR) 217C,2170,2160
0191           2160 WRITE (3,217) JCONT(1,I),JCONT(2,I),JCONT(3,I)
0192            217 FORMAT (I16,I10,I9)
0193           2170 NCONT = NCONT-1
0194                IF (I-NCONT) 2171,2171,220
0195           2171 DO 2172 J = I,NCONT
0196                JCONT(1,J) = JCONT(1,J+1)
0197                JCONT(2,J) =JCCNT(2,J+1)
0198           2172 JCONT(3,J) = JCONT(3,J+1)
0199                GO TO 215
0200            220 CONTINUE
              C
              C     END OF SCAN LOCP
0201            225 CONTINUE
              C
              C     PRINT REMAINING CONTINUITIES IF MORE THAN INPUT QTY
0202            230 DO 235 K = 1,NCONT
0203                IF (JCONT(2,K)-NCNTPR) 235,235,232
0204            232 WRITE (3,217) JCONT(1,K),JCONT(2,K),JCONT(3,K)
0205            235 CONTINUE
              C
0206            899 IF (NARRPR) 90C,900,940
              C     PRINT OUT ALL SSA LOCATIONS
0207            900 WRITE (3,901)
0208            901 FORMAT(1H1//10X,13HSSA LOCATIONS///16X,12HELE     SCANS/
                   115X,5HMENTS,9X,1H1,9X,1H2,9X,1H3,9X,1H4,9X,1H5,9X,1H6,9X,1H7,
                   29X,1H8,9X,1H9/
                   320X,100H1234567890123456789012345678901234567890123456789012345678
                   4901234567890123456789012345678901234567890//)
              C     SET UP A LINE FOR EACH ELEMENT
0209            908 DO 909  I = 1,NSCAN
```

```
0210          909 NPLACE(I) = 1
0211          910 DO 935 I = 1,NARRAY
            C         LINE TO SPACES
0212              DO 912 J = 1,100
0213          912 PLINE(J) = SPACE
            C         SEARCH WITHIN EACH SCAN
0214          914 DO 922 KSCAN = 1,NSCAN
0215              K = NPLACE(KSCAN)
0216              IF (JSSA(K,KS[illegible]) I) 922,918,922
0217          918 PLINE(KSCAN) [illegible]AR
0218              NPLACE(KSCAN) [illegible]ACE(KSCAN) + 1
0219          922 CONTINUE
            C         PRINT LINE
0220          930 WRITE (3,931) I,(PLINE(J), J = 1,NSCAN)
0221          931 FORMAT (I18,2X,100A1)
0222          935 CONTINUE
            C
0223          940 CONTINUE
0224              GO TO 10
0225          999 CONTINUE
0226              CALL EXIT
0227              END
```

```
C      RANDOM NUMBER GENERATOR SUBROUTINE
C
        SUBROUTINE RANDU(IX,IY,YFL)
        IY = IX*65539
        IF (IY) 5,6,6
      5 IY = IY+2147483647+1
      6 YFL = IY
        YFL = YFL*.4656613E-9
        RETURN
        END
```

APPENDIX B

MICROPROCESSOR IMPLEMENTATION

```
ISIS-II PL/M-80 V2.0 COMPILATION OF MODULE MATX
OBJECT MODULE PLACED IN :F1:ARRAY.OBJ
COMPILER INVOKED BY:  PLM80 :F1.ARRAY.SRC   PRINT(:LP:)


            /*

                        **********************

                        PROGRAM NAME 'ARRAY'

                           DATE 3/11/77

                  **************************************


               EXTERNAL PROCEDURE DECLARATIONS FOR ISIS-II SYSTEM CALLS

            */

  1         MATX:  DO;
  2    1    OPEN:
                PROCEDURE(AFT,FILE,ACCESS,MODE,STATUS)EXTERNAL;
  3    2        DECLARE(AFT,FILE,ACCESS,MODE,STATUS)ADDRESS;
  4    2        END OPEN;
  5    1    WRITE:
                PROCEDURE(AFT,BUFFER,COUNT,STATUS)EXTERNAL;
  6    2        DECLARE(AFT,BUFFER,COUNT,STATUS)ADDRESS;
  7    2        END WRITE;
  8    1    READ:
                PROCEDURE(AFT,BUFFER,COUNT,ACTUAL,STATUS)EXTERNAL;
  9    2        DECLARE(AFT,BUFFER,COUNT,ACTUAL,STATUS)ADDRESS;
 10    2        END READ;
 11    1    EXIT:
                PROCEDURE EXTERNAL;
 12    2        END EXIT;
            /*

               DECLARE VARIABLES

            */
 13    1    DECLARE BUFFER(128) BYTE;
 14    1    DECLARE (STATUS,ACTCNT,AFTDSK,AFTIN,T,P,A) ADDRESS;
 15    1    DECLARE LETTER$X LITERALLY '58H';
 16    1    DECLARE SP LITERALLY '20H';
 17    1    DECLARE CRLF LITERALLY '0AH,0DH';
 18    1    DECLARE (I, ,N,L,K,D,E,F) BYTE;
 19    1    DECLARE FF LITERALLY '0CH';
 20    1    DECLARE LP(5)BYTE DATA(':LP:  ');
 21    1    DECLARE ROW(50) STRUCTURE (COL(70) BYTE);

            /*
               GENERATE AN ARRAY 70 ACROSS AND 50 DOWN
            */
 22    1    GEN:  DO;
 23    2       DOWN:    DO J=0 TO 49;
 24    3          ACROSS:    DO I=0 TO 69;
```

```
25    4                                   ROW(J). COL(I
26    4                               END;
27    3                       END;
28    2                 END;

                  /*
                     THE OPERATOR SHALL INPUT A 5-DIGIT NUMBER FOR THE RANDOM NUMBE
                     GENERATOR AND A 3-DIGIT NUMBER FOR THE PERCENT OF NOISE.
                  */
29    1           LOAD:  DO;
30    2           CALL WRITE (0,.(CRLF,'INPUT A 5-DIGIT NUMBER AND
                  RETURN',CRLF),37,.STATUS);
31    2           CALL READ (1,.BUFFER,6,.ACTCNT,.STATUS);
32    2           CALL READ (1,.BUFFER,6,.ACTCNT,.STATUS);
33    2           A=BUFFER(5)-30H+(BUFFER(4)-30H)*10+(BUFFER(3)-30H)*100
                  +(BUFFER(2)-30H)*1000
                  +(BUFFER(1)-30H)*10000+(BUFFER(0)-30H)*100000;
34    2           CALL WRITE (0,.(CRLF,'INPUT A 3-DIGIT NUMBER,REPRESENTING
                   THE PERCENT OF BACKGROUND NOISE,AND A RETURN',CRLF),84,.STATUS)
35    2           CALL READ (1,.BUFFER,3,.ACTCNT,.STATUS);
36    2           CALL READ (1,.BUFFER,3,.ACTCNT,.STATUS);
37    2                   D=BUFFER(2);
38    2                   E=BUFFER(1);
39    2                   F=BUFFER(0);
40    2           N=BUFFER(2)-30H+(BUFFER(1)-30H)*10+(BUFFER(0)-30H)*100;
41    2                P=65535-655*N;
42    2           END LOAD;
                  /*
                      RANDOM NUMBER GENERATOR
                  */
43    1               T=0;
44    1           RANDOM:  DO WHILE T<3500;
45    2                  JX:       DO J=0 TO 49;
46    3                     IX:        DO I=0 TO 69;

47    4                  IF A>P THEN
48    4                              DO;
49    5                                 ROW(J).COL(I)=(LETTER$X);
50    5                              END;
                                 ELSE
51    4                              DO;
52    5                                 ROW(J).COL(I)=(SP);
53    5                              END;
54    4                   A=(2053*A+13849) MOD 65535;
55    4                   T=T+1;
56    4                            END IX;
57    3                      END JX;
58    2           END RANDOM;

                  /*
                     INPUT WIRE
                  */
59    1               J=1;
60    1           CALL WRITE (0,.(CRLF,CRLF),4,.STATUS);

61    1           CALL WRITE (0,.('INPUT WIRE',CRLF),12,.STATUS);
62    1           CALL WRITE (0,.('NOTE: ROW>70 TERMINATES
                   INPUT WIRE',CRLF),36,.STATUS);
63    1           WIRE:  DO WHILE J<70,
```

```
 64   2      CALL WRITE (0,.(CRLF,CRLF),4,.STATUS);
 65   2      CALL WRITE (0,.('INPUT A 2-DIGIT NUMBER FOR ROW
              LOCATION AND RETURN',CRLF),52,.STATUS);
 66   2      CALL READ (1,.BUFFER,2,.ACTCNT,.STATUS);
 67   2      CALL READ (1,.BUFFER,2,.ACTCNT,.STATUS);

 68   2      J=(BUFFER(1)-30H+(BUFFER(0)-30H)*10)-1;
 69   2         IF J<70 THEN
 70   2               DO;

 71   3      CALL WRITE (0,.('INPUT A 2-DIGIT NUMBER FOR COLUMN
              LOCATION AND RETURN',CRLF),55,.STATUS);

 72   3      CALL READ (1,.BUFFER,2,.ACTCNT,.STATUS);
 73   3      CALL READ (1,.BUFFER,2,.ACTCNT,.STATUS);

 74   3      I=(BUFFER(1)-30H+(BUFFER(0)-30H)*10)-1;

 75   3         ROW(J).COL(I)=(LETTER$N);
 76   3         END;
 77   2      END WIRE;

 78   1      J=0;
 79   1      L=1;
 80   1      K=0;
             /*
                LINE PRINTER PRINTOUT
             */
 81   1      CALL OPEN (.AFTIN,.LP,2,0,.STATUS);
 82   1      CALL WRITE (AFTIN,.(FF,CRLF),3,.STATUS);
 83   1      CALL WRITE (AFTIN,.('                                   SITE
              ARRAY WITH '),39,.STATUS);
 84   1      CALL WRITE (AFTIN,.F,1,.STATUS);
 85   1      CALL WRITE (AFTIN,.E,1,.STATUS);
 86   1      CALL WRITE (AFTIN,.D,1,.STATUS);
 87   1      CALL WRITE (AFTIN,.(' PERCENT OF BACKGROUND NOISE
             ',CRLF,CRLF,CRLF),35,.STATUS);
 88   1      CALL WRITE (AFTIN,.('                              1          2         3
                     4          5          6           7',CRLF),82,.STATUS);
 89   1      CALL WRITE (AFTIN,.('                  123456789012345678901234567890
             1234567890123456789012345678901234567890'),86,.STATUS);
 90   1      CALL WRITE (AFTIN,.(CRLF,CRLF,CRLF),6,.STATUS);
 91   1      RIT:  DO WHILE J<50;
 92   2        K=K+30H;
 93   2        L=L+30H;
 94   2      CALL WRITE (AFTIN,.('     '),5,.STATUS);
 95   2      CALL WRITE (AFTIN,.K,1,.STATUS);
 96   2      CALL WRITE (AFTIN,.L,1,.STATUS);
 97   2      CALL WRITE (AFTIN,.('   '),3,.STATUS);
 98   2      CALL WRITE (AFTIN,.ROW(J).COL,70,.STATUS);
 99   2      CALL WRITE (AFTIN,.(CRLF),2,.STATUS);
100   2           K=K-30H;
101   2             L=L-30H;
102   2             J=J+1;
103   2             L=L+1;
104   2                     IF L=10  THEN
105   2                                DO;
106   3                                   L=0;
107   3                                   K=K+1;
108   3                                END;
```

```
 109   2      END RIT;
              /*
                 DISK FILE NAME
              */
 110   1      CALL WRITE (0,.(CRLF,CRLF),4,.STATUS);
 111   1      CALL WRITE (0,.('INPUT FILE NAME AND RETURN',CRLF),28,.STATUS);
 112   1      CALL READ (1,.BUFFER,128,.ACTCNT,.STATUS);
 113   1      CALL READ (1,.BUFFER,128,.ACTCNT,.STATUS);

              /*
                MAKE A FILE
              */
 114   1      CALL OPEN (.AFTDSK,.BUFFER,2,0,.STATUS);
 115   1      NEWFIL:   DO;
 116   2                         DO  I=0 TO 69;
 117   3                            DO  J=0 TO 49;

 118   4      CALL WRITE (AFTDSK,.ROW(J).COL(I),1,.STATUS);
 119   4                            END;
 120   3                         END;
 121   2      END NEWFIL;
 122   1      END MATX;


MODULE INFORMATION:

     CODE AREA SIZE     = 07E6H   2022D
     VARIABLE AREA SIZE = 0E42H   3650D
     MAXIMUM STACK SIZE = 0008H      8D
     196 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V2.0 COMPILATION OF MODULE DET
OBJECT MODULE PLACED IN :F1:DETECT.OBJ
COMPILER INVOKED BY:  PLM80 :F1:DETECT.SRC   PRINT(:LP:)


              /*
                                     **********************

                                     PROGRAM NAME 'DETECT'

                                         DATE 3/15/77

                              **********************************

                 EXTERNAL PROCEDURE DECLARATIONS FOR ISIS-II SYSTEM CALLS

              */

  1           DET:    DO;


  2    1      OPEN:
                  PROCEDURE (AFT,FILE,ACCESS,MODE,STATUS) EXTERNAL;
  3    2          DECLARE (AFT,FILE,ACCESS,MODE,STATUS) ADDRESS;
  4    2          END OPEN;
  5    1      WRITE:
                  PROCEDURE (AFT,BUFFER,COUNT,STATUS) EXTERNAL;
  6    2          DECLARE (AFT,BUFFER,COUNT,STATUS) ADDRESS;
  7    2          END WRITE;
  8    1      READ:
                  PROCEDURE (AFT,BUFFER,COUNT,ACTUAL,STATUS) EXTERNAL;
  9    2          DECLARE (AFT,BUFFER,COUNT,ACTUAL,STATUS) ADDRESS;
 10    2          END READ;

              /*
                 DECLARE VARIABLES
              */

 11    1      DECLARE FF LITERALLY '0CH';
 12    1      DECLARE LP(3)BYTE DATA(':LP: ');
 13    1      DECLARE CRLF LITERALLY '0AH,0DH';
 14    1      DECLARE BUFFER(128) BYTE;
 15    1      DECLARE (STATUS,ACTCNT,AFTIN,AFTDSK) ADDRESS;
 16    1      DECLARE (L,C,N) BYTE;
 17    1      DECLARE LINE(70)STRUCTURE(CELL(58)BYTE);
 18    1      DECLARE LL LITERALLY '0EH';
 19    1      DECLARE E(2) BYTE;
 20    1      DECLARE S(2) BYTE;


              /*
                 INPUT SCAN OF 50 ELEMENTS
              */

 21    1      GET:
                   PROCEDURE;
 22    2           DO C=4 TO 53;
 23    3           CALL READ(AFTDSK, BUFFER,1, ACTCNT, STATUS);
```

```
24   3          IF BUFFER(0)=58H  THEN
25   3          DO;
26   4          LINE(L).CELL(C)=1;
27   4          END;
                ELSE
28   3          DO;
29   4          LINE(L).CELL(C)=0;
30   4          END;
31   3          END;
32   2          END GET;

           /*
              ADD FOUR ELEMENTS AT BEGINING AND END OF SCAN
           */

33   1     START:  DO;
34   2     DO L=0 TO 69;
35   3        DO C=0 TO 3;
36   4           LINE(L).CELL(C)=0;
37   4        END;
38   3        DO C=54 TO 57;
39   4           LINE(L).CELL(C)=0;
40   4        END;
41   3     END;

           /*
              INPUT DETERMINATION CRITERION AND FILE NAME
           */

42   2     CALL WRITE (0,.(CRLF,CRLF,'INPUT A 2-DIGIT
            DETERMINATION CRITERION AND RETURN',CRLF),56,.STATUS);

43   2     CALL READ (1,.BUFFER,2,.ACTCNT,.STATUS);
44   2     CALL READ (1,.BUFFER,2,.ACTCNT,.STATUS);
45   2        N=BUFFER(1)-30H+(BUFFER(0)-30H)*10;

46   2     CALL WRITE (0,.('INPUT FILE NAME AND RETURN',CRLF),28,.STATUS);
47   2     CALL READ (1,.BUFFER,128,.ACTCNT,.STATUS);
48   2     CALL READ (1,.BUFFER,128,.ACTCNT,.STATUS);
49   2     CALL OPEN (.AFTDSK,.BUFFER,1,0,.STATUS);


           /*
              LOCATE WIRE
           */

50   2     BEGIN:  DO L=0 TO 69;
51   3               CALL GET;
52   3                DO C=4 TO 53;
53   4                 IF LINE(L).CELL(C)=1 THEN
54   4                 DO;

55   5                IF L>0 THEN
56   5                    DO;
57   6                     IF LINE(L-1).CELL(C+1)>0 THEN
58   6                     DO;
59   7                      LINE(L).CELL(C)=LINE(L-1).CELL(C+1)+1;
60   7                      GOTO TEST;
61   7                     END;
62   6                     IF LINE(L-1).CELL(C)>0 THEN
```

```
6                      DO;
7                       LINE(L).CELL(C)=LINE(L-1).CELL(C)+1;
7                       GOTO TEST;
7                      END;
6                      IF LINE(L-1).CELL(C-1)>0 THEN
6                      DO;
7                       LINE(L).CELL(C)=LINE(L-1).CELL(C-1)+1;
7                       GOTO TEST;
7                      END;
6                     END;

5              IF L>1 THEN
5                 DO;
6                    IF LINE(L-2).CELL(C+2)>0 THEN
6                    DO;
7                     LINE(L).CELL(C)=LINE(L-2).CELL(C+2)+1;
7                     GOTO TEST;
7                    END;
6                    IF LINE(L-2).CELL(C+1)>0 THEN
6                    DO;
7                     LINE(L).CELL(C)=LINE(L-2).CELL(C+1)+1;
7                     GOTO TEST;
7                    END;
6                    IF LINE(L-2).CELL(C)>0 THEN
6                    DO;
7                     LINE(L).CELL(C)=LINE(L-2).CELL(C)+1;
7                     GOTO TEST;
7                    END;
6                    IF LINE(L-2).CELL(C-1)>0 THEN
6                    DO;
7                     LINE(L).CELL(C)=LINE(L-2).CELL(C-1)+1;
7                     GOTO TEST;
7                    END;
6                    IF LINE(L-2).CELL(C-2)>0 THEN
6                    DO;
7                     LINE(L).CELL(C)=LINE(L-2).CELL(C-2)+1;
7                     GOTO TEST;
7                    END;
6                 END;

5              IF L>2 THEN
5                   DO;
6                    IF LINE(L-3).CELL(C+4)>0 THEN
6                    DO;
7                     LINE(L).CELL(C)=LINE(L-3).CELL(C+4)+1;
7                     GOTO TEST;
7                    END;
6                    IF LINE(L-3).CELL(C+3)>0 THEN
6                    DO;
7                     LINE(L).CELL(C)=LINE(L-3).CELL(C+3)+1;
7                     GOTO TEST;
7                    END;
6                    IF LINE(L-3).CELL(C+2)>0 THEN
6                    DO;
7                     LINE(L).CELL(C)=LINE(L-3).CELL(C+2)+1;
7                     GOTO TEST;
7                    END;
6                    IF LINE(L-3).CELL(C+1)>0 THEN
6                    DO;
7                     LINE(L).CELL(C)=LINE(L-3).CELL(C+1)+1;
```

```
121   7                         GOTO TEST·
122   7                        END;
123   6                        IF LINE(L-3).CELL(C)>0 THEN
124   6                        DO;
125   7                         LINE(L).CELL(C)=LINE(L-3).CELL(C)+1;
126   7                         GOTO TEST;
127   7                        END;
128   6                        IF LINE(L-3).CELL(C-1)>0 THEN
129   6                        DO;
130   7                         LINE(L).CELL(C)=LINE(L-3).CELL(C-1)+1;
131   7                         GOTO TEST;
132   7                        END;
133   6                        IF LINE(L-3).CELL(C-2)>0 THEN
134   6                        DO;
135   7                         LINE(L).CELL(C)=LINE(L-3).CELL(C-2)+1;
136   7                         GOTO TEST;
137   7                        END;
138   6                        IF LINE(L-3).CELL(C-3)>0 THEN
139   6                        DO;
140   7                         LINE(L) .CELL(C)=LINE(L-3).CELL(C-3)+1;
141   7                         GOTO TEST;
142   7                        END;
143   6                        IF LINE(L-3).CELL(C-4)>0 THEN
144   6                        DO;
145   7                         LINE(L).CELL(C)=LINE(L-3).CELL(C-4)+1;
146   7                         GOTO TEST;
147   7                        END;
148   6                       END;


149   5                   END;

150   4          TEST:    DO;
151   5                   IF LINE(L).CELL(C)=N THEN
152   5                      DO;
153   6                        GOTO ALARM;
154   6                      END;
155   5                   END TEST;

156   4                  END;

157   3       END BEGIN;

158   2       CALL OPEN (.AFTIN,.LP,2,0,.STATUS);
159   2       CALL WRITE (AFTIN,.(FF),2,.STATUS);
160   2       CALL WRITE (AFTIN,.('         NO WIRE DETERMINATION IN
               THIS SCENE',CRLF),47,.STATUS);


161   2       GOTO STOP;

162   2       ALARM:  DO;

163   3            C=C-3;
164   3            E(0)=C/10;
165   3            E(1)=C-(E(0)*10);
166   3            E(1)=E(1)+30H;
167   3            E(0)=E(0)+30H;
```

```
168   3              L=L+1;
169   3              S(0)=L/10;
170   3              S(1)=L-(S(0)*10);
171   3              S(1)=S(1)+30H;
172   3              S(0)=S(0)+30H;


173   3        CALL OPEN (.AFTIN,.LP,2,0, STATUS);
174   3        CALL WRITE (AFTIN,.(FF),2,.STATUS);
175   3        CALL WRITE (AFTIN,.(LL,'                    WOW'),22, STATUS);
176   3        CALL WRITE (AFTIN,.(CRLF),2,.STATUS);
177   3        CALL WRITE (AFTIN,.(LL,'                   ALERT'),23, STATUS);
178   3        CALL WRITE (AFTIN,.(CRLF,CRLF,CRLF,CRLF,
               '                   WIRE DETERMINATION MADE AT ELEMENT '),
               ,62,.STATUS);
179   3        CALL WRITE (AFTIN,.E,2,.STATUS);
180   3        CALL WRITE (AFTIN,.(',SCAN '),6, STATUS);
181   3        CALL WRITE (AFTIN,.S,2, STATUS);
182   3        CALL WRITE (AFTIN,.(CRLF,CRLF),4, STATUS);

183   3                END ALARM;


184   2        END START;


185   1        STOP:  DO;
186   2               END;

187   1        END DET;
```

```
MODULE INFORMATION:

     CODE AREA SIZE     = 0ABCH   2748D
     VARIABLE AREA SIZE = 106BH   4203D
     MAXIMUM STACK SIZE = 000AH     10D
     264 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

APPENDIX C

NON-CONTIGUOUS SCAN PROGRAM

NARRAY	Number of elements in array.
NTRUE	Number of true wire-like inputs.
NSCAN	Number of scans to be processed.
NCNTPR	Criterion for wire-determination printing.
NARRPR	Single site map print option; may is suppressed if NARRPR $\neq$ 0
NDRAW	Extra input option; Additional individual single site input will be processed if NDRAW $\neq$ 0
PBACK	Percentage of background random single sites.
IWBG1 IWBG2	Element size of large window in defining a wire possibility from 1st. occurance scan to the next scan.
IWSM1 IWSM2	Element size of small window in defining a wire continuity beyond the second occurance.
IWSMA IWSMB	Small window sizes modified for skipped scans and and horizontal slope.
NCONT	Number of entries at any time in the continuity table.
NPOSS	Number of entries at any time in the possibilites table.

IEXP	Expected value for the single site wire on the next scan. For a second occurance, IEXP is taken as the value of the 1st. occurance; subsequently it is found by linear extrapolation of the 2 most recent occurances.
IDELTA	Expected difference in element value from one scan to the next.
MULT	Multiplication factor for previous skipped scans.

NTRUE1 = NTRUE - 1

NAR1 = NARRAY - 1

NCONT1 = NCONT - 1

CONTINUITIES TABLE

JCONT (1, KCONT)	Element value for next-to-last occurance of this continuity.
JCONT (2, KCONT)	Element value for last occurance of this continuity.
JCONT (3, KCONT)	Scan number of last occurance.
JCONT (4, KCONT)	Total number of occurances for this continuity.
JCONT (5, KCONT)	Usage indicator. All continuities start each scan with a value of zero. If the continuity is continued, the value is set to 1, a new continuity is entered and this value eliminated at the end of the scan.
JCONT (6, KCONT)	A multiplication factor for computing next delta value when a scan has been shipped.

POSSIBILITIES TABLE

JPOSS (1, KPOSS) element value

JPOSS (2, KPOSS) scan number for element

TRUE SINGLE SITE INPUT

ITRUE element number initiating wire-like obstacle.

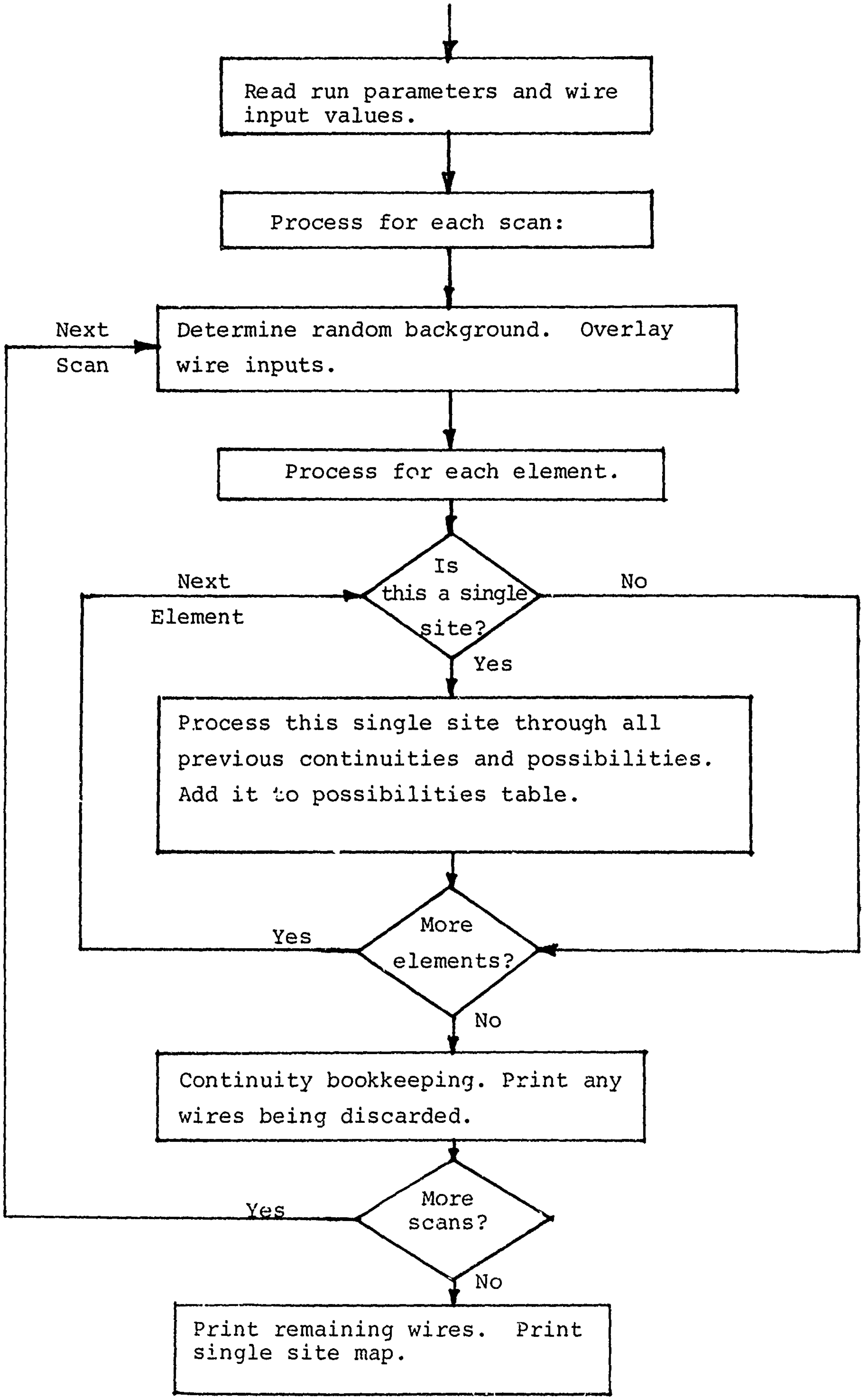
ITDEL data value from one scan to the next.

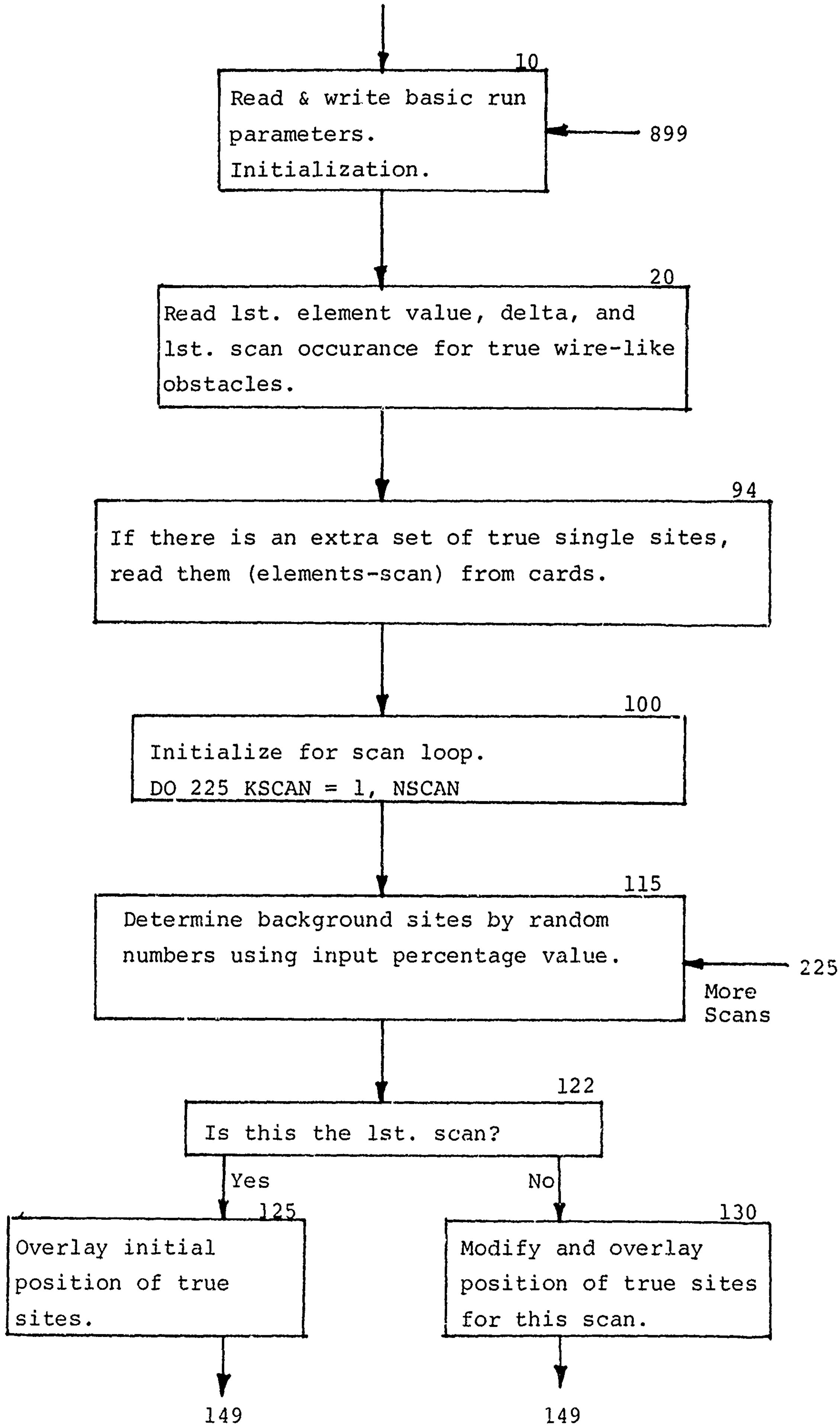
IFIRST scan number on which the wire-like obstacle first appears.

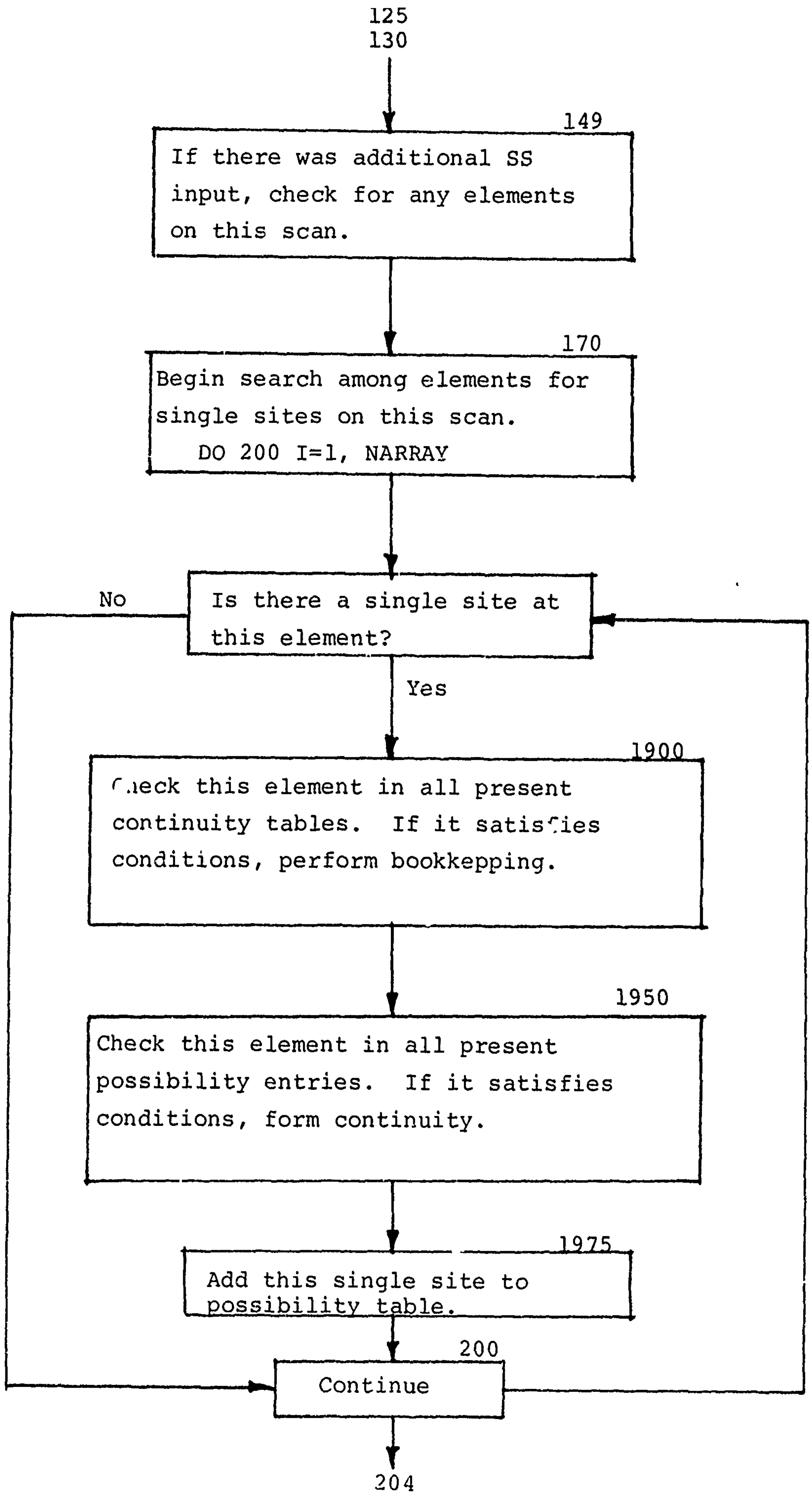
ADDITIONAL INPUT WHEN NDRAW $\neq$ 0

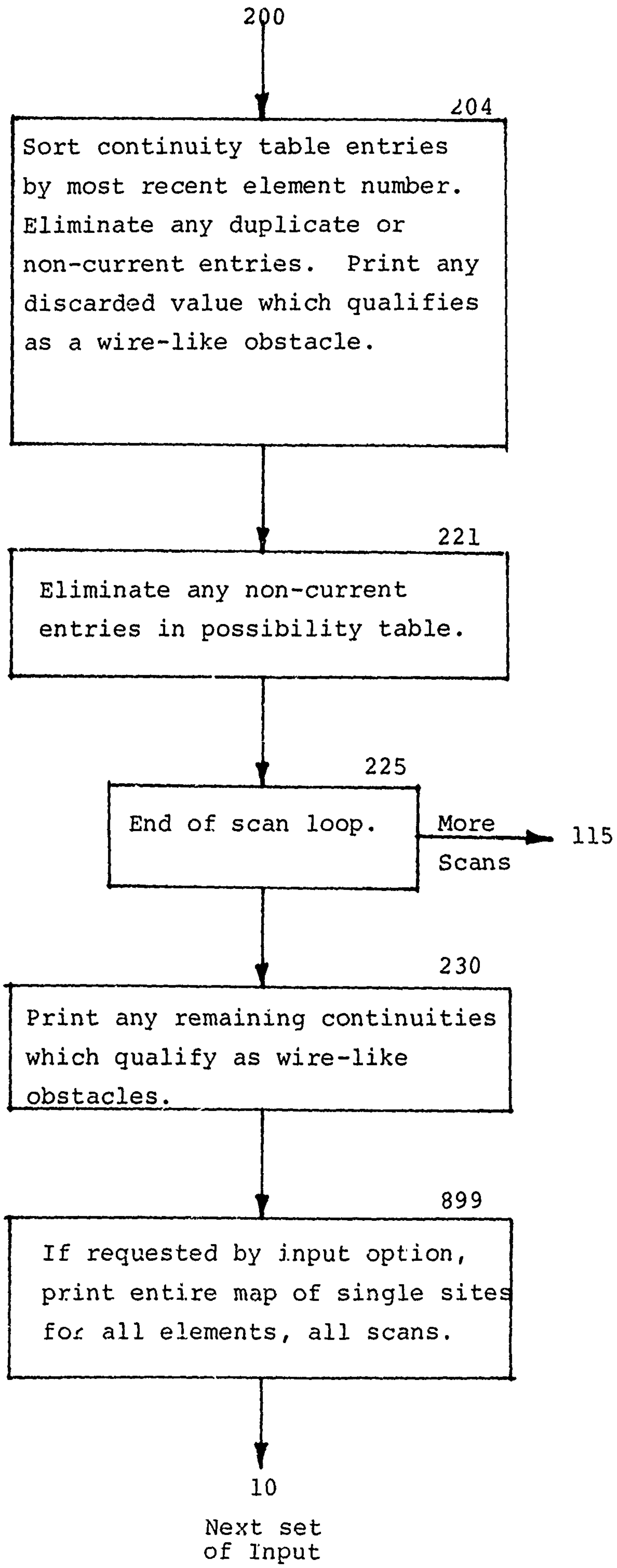
ISCAN scans on which single sites are placed.

IELEM elements on which single sites are placed.


Read run parameters and wire input values.
Process for each scan:
Next Scan
Determine random background. Overlay wire inputs.
Process for each element.
Next Element
Is this a single site?
No
Yes
Process this single site through all previous continuities and possibilities. Add it to possibilities table.
Yes
More elements?
No
Continuity bookkeeping. Print any wires being discarded.
Yes
More scans?
No
Print remaining wires. Print single site map.

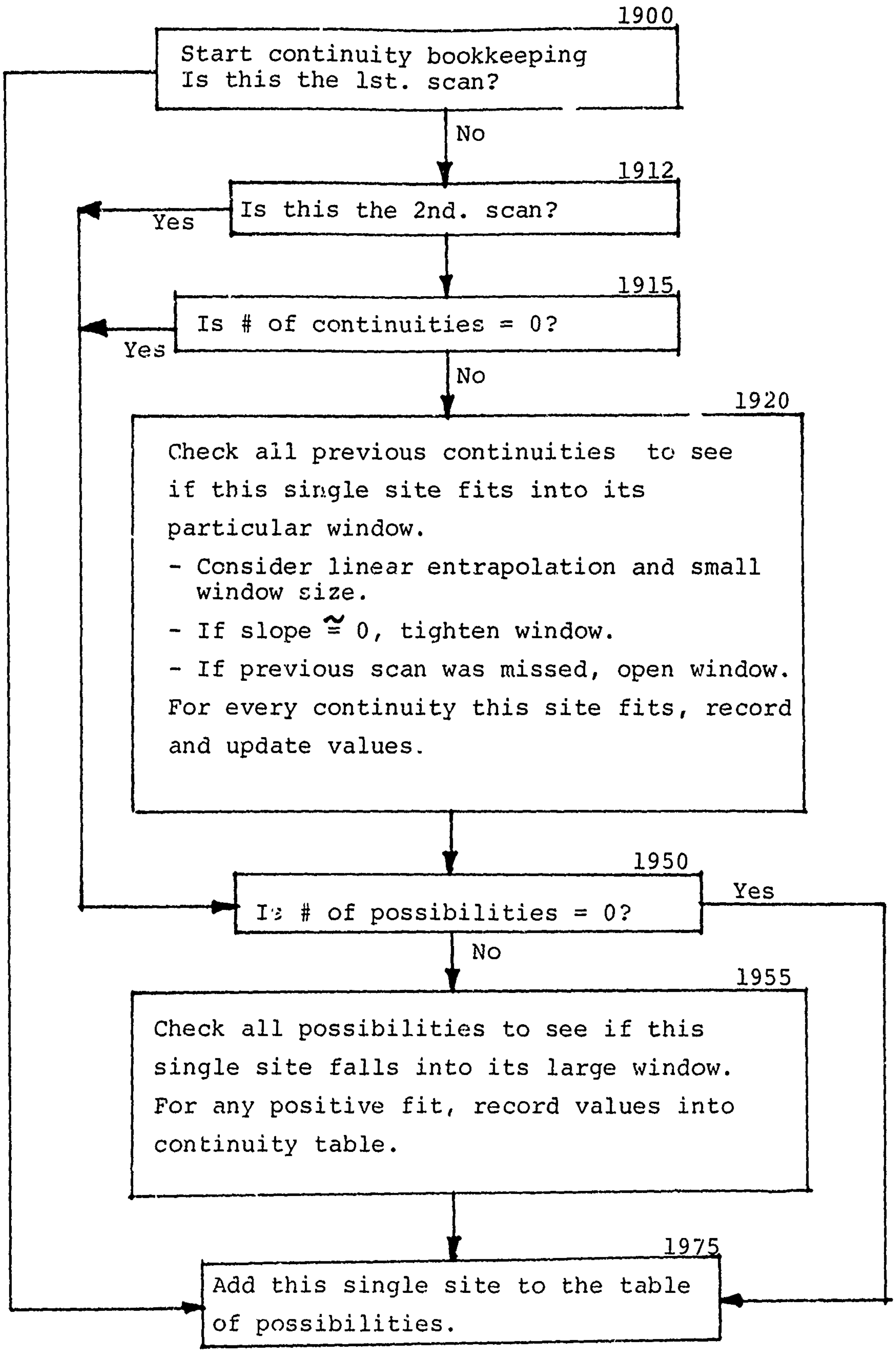

10
Read & write basic run parameters. Initialization.
899
20
Read lst. element value, delta, and lst. scan occurance for true wire-like obstacles.
94
If there is an extra set of true single sites, read them (elements-scan) from cards.
100
Initialize for scan loop.
DO 225 KSCAN = 1, NSCAN
115
Determine background sites by random numbers using input percentage value.
225
More Scans
122
Is this the lst. scan?
Yes
No
125
Overlay initial position of true sites.
130
Modify and overlay position of true sites for this scan.
149
149


125
130
149
If there was additional SS input, check for any elements on this scan.
170
Begin search among elements for single sites on this scan.
DO 200 I=1, NARRAY
Is there a single site at this element?
No
Yes
1900
Check this element in all present continuity tables. If it satisfies conditions, perform bookkepping.
1950
Check this element in all present possibility entries. If it satisfies conditions, form continuity.
1975
Add this single site to possibility table.
200
Continue
204

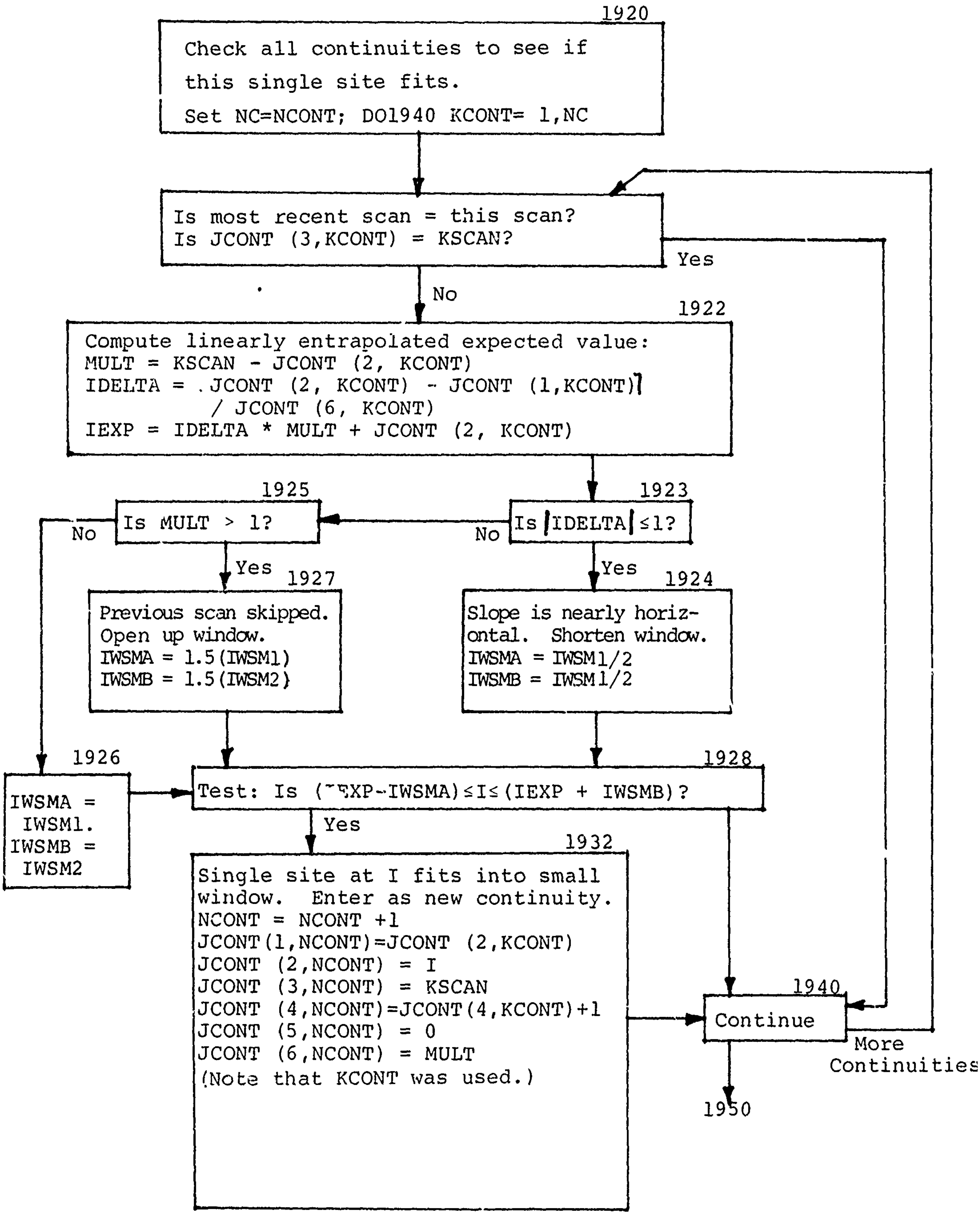

200
204
Sort continuity table entries by most recent element number. Eliminate any duplicate or non-current entries. Print any discarded value which qualifies as a wire-like obstacle.
221
Eliminate any non-current entries in possibility table.
225
End of scan loop.
More Scans
115
230
Print any remaining continuities which qualify as wire-like obstacles.
899
If requested by input option, print entire map of single sites for all elements, all scans.
10
Next set of Input

Detail of Boxes 1900-1975

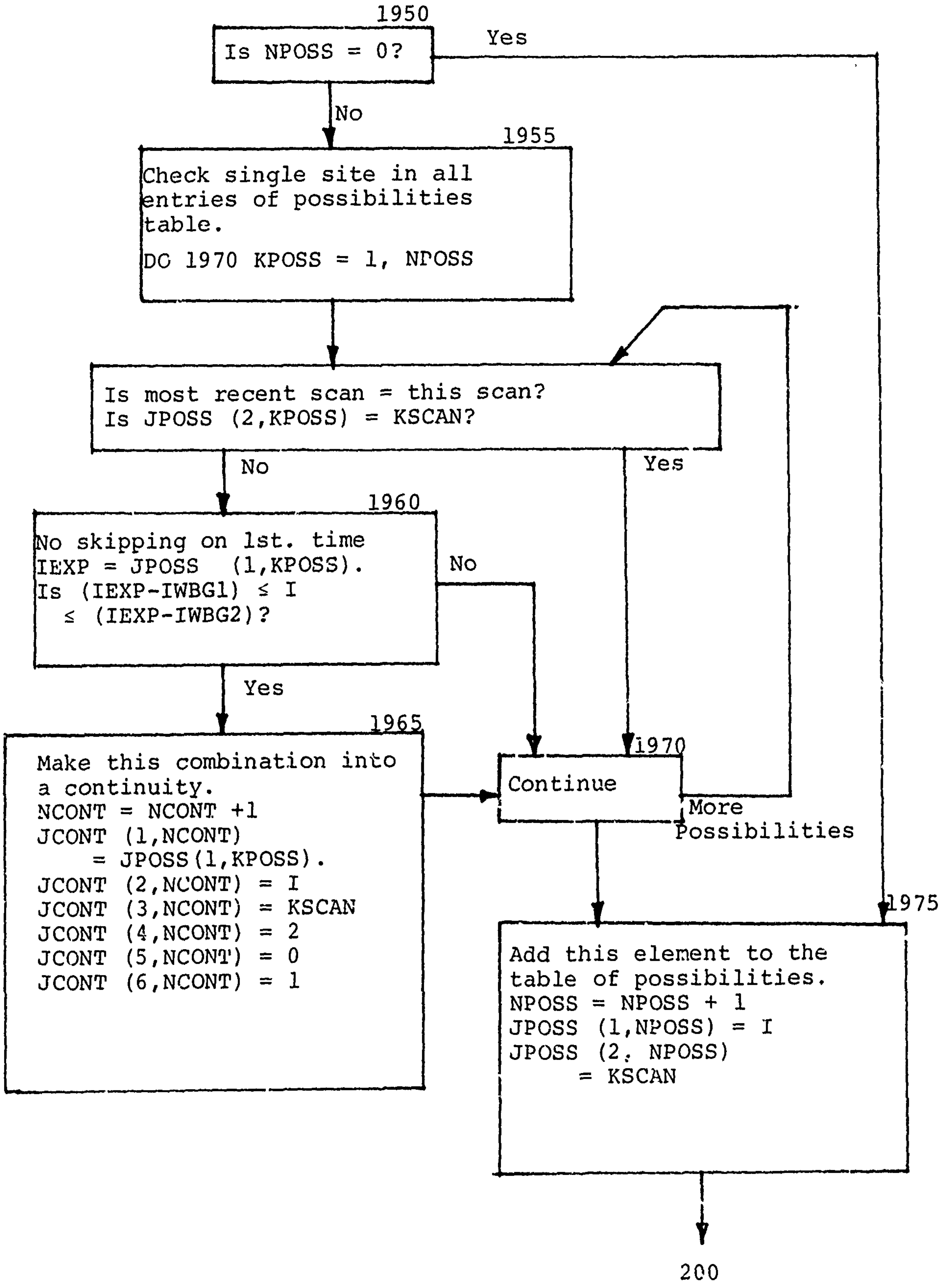
(Have single site)



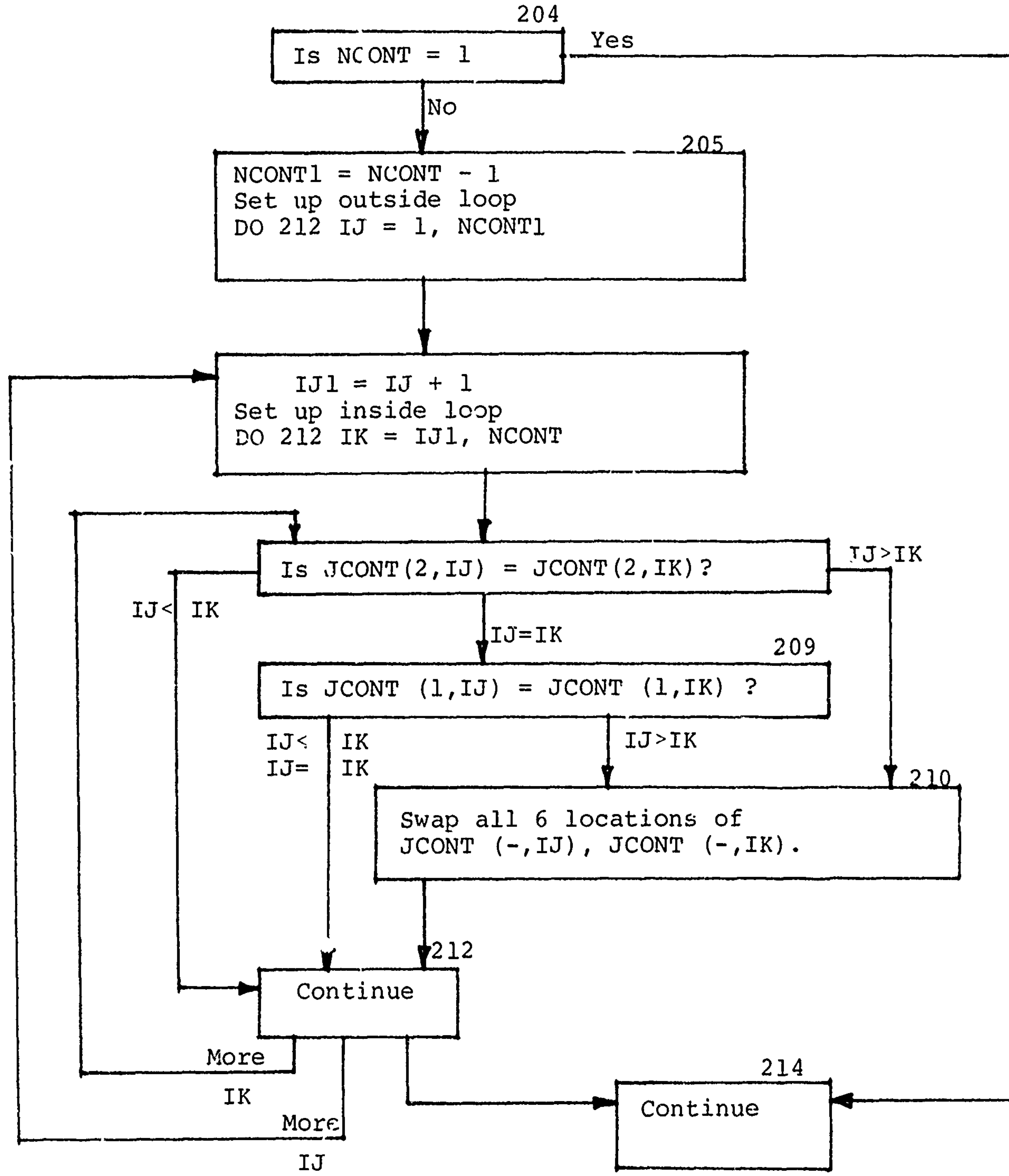
Detail of Box 1920
(Check continuities)



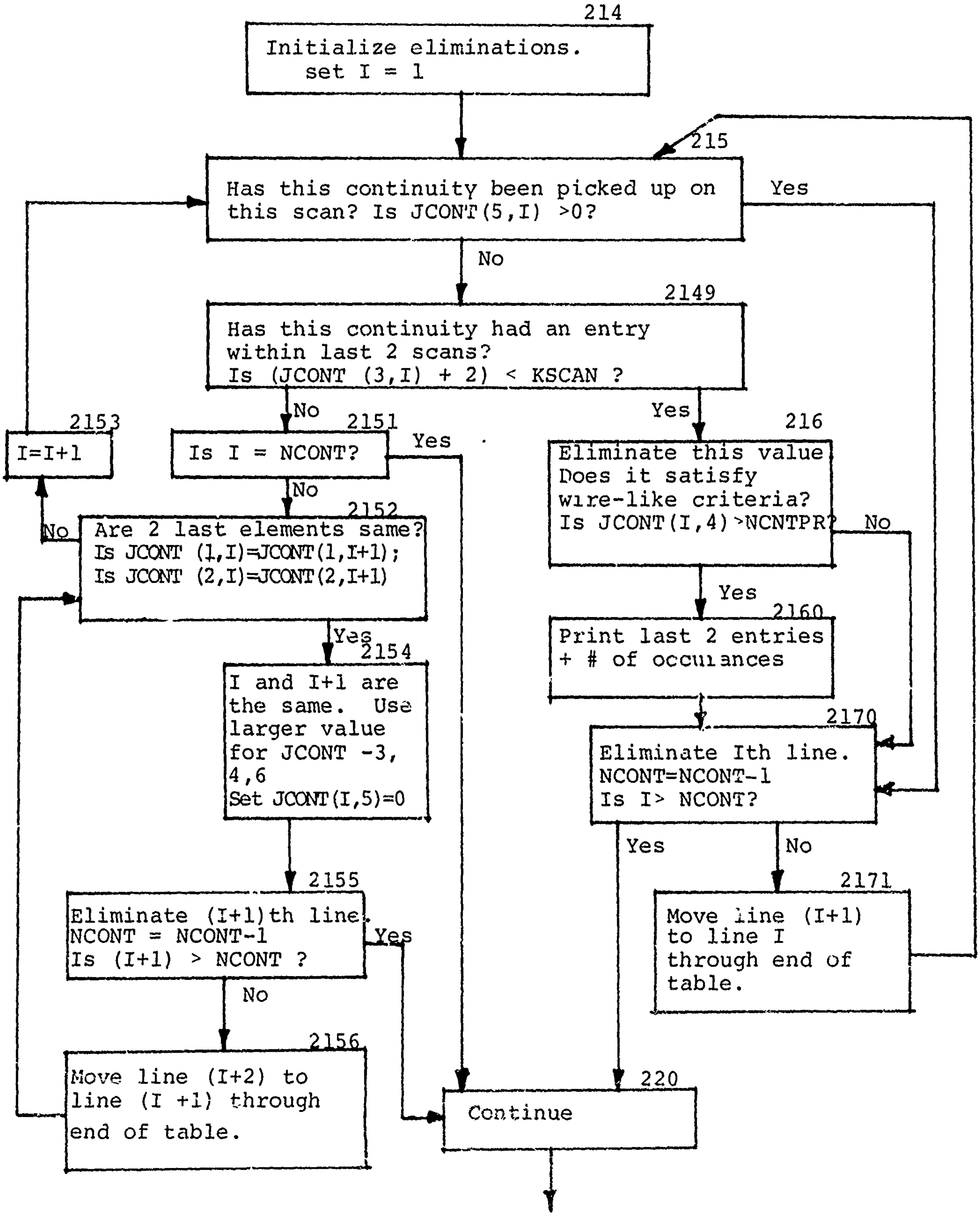
Detail of boxes 1950-1975



Detail of continuity sorting



Detail of continuity elimination
(duplicates and non-current)



```
                C      SINGLE SITE PREDICTION
                C
 0001                  DIMENSION ITRUE(10), IARRAY(1729), JSSA(200,100),RUNID(20)
 0002                  DIMENSION PLINE(100), NSSA(100),NPLACE(100),JCONT(6,200)
 0003                  DIMENSION ISCAN(160),IELEM(160),ITDEL(10),JPOSS(2,200),IFIRST(10)
 0004                  DATA SPACE/' '/,STAR/'X'/
                C
                C      READ INPUT PARAMETERS
 0005               10 READ (1,11,END=999) RUNID
 0006               11 FORMAT (20A4)
 0007                  READ (1,12) IX, NARRAY,NTRUE,NSCAN,NCNTPR,NARRPR,NDRAW,PBACK
 0008               12 FORMAT (7I5,F10.0)
                C      READ WINDOW SIZES
 0009               13 READ (1,12)  IWBG1,IWBG2,IWSM1,IWSM2
 0010                  WRITE (3,15) RUNID,NARRAY,NSCAN,NTRUE,PBACK,NCNTPR,IX
 0011               15 FORMAT (1H1///10X,20A4///10X,I6,18H = LENGTH OF ARRAY//
                      110X,I6,18H = NUMBER OF SCANS//10X,I6,20H = SITES TO BE INPUT//
                      210X,F6.1,31H = PERCENTAGE RANDOM BACKGROUND//
                      310X,I6,22H = CONTINUITY CRITERIA//
                      410X,I6,28H = RANDOM NUMBER INITIALIZER//)
 0012               16 DO 17 I = 1,200
 0013                  DO 17 K = 1,100
 0014               17 JSSA(I,K) = 0
 0015                  IARRAY(NARRAY+1) = 0
 0016                  NAR1 = NARRAY - 1
 0017               18 WRITE (3,19) IWBG1,IWBG2,IWSM1,IWSM2
 0018               19 FORMAT (//10X,'LARGE AND SMALL WINDOW SIZES'//10X,4I5)
                C
                C      READ TRUE SITES AND DELTAS
 0019               20 IF (NTRUE) 70,70,22
 0020               22 DO 25 J = 1,NTRUE
 0021                  READ (1,24) ITRUE(J),ITDEL(J),IFIRST(J)
 0022               24 FORMAT (3I5)
 0023               25 CONTINUE
 0024                  IF (NTRUE-1) 90,90,28
 0025               28 NTRUE1 = NTRUE - 1
 0026               30 DO 36 I = 1,NTRUE1
 0027                  I1 = I + 1
 0028               32 DO 36 J = I1,NTRUE
 0029                  IF (ITRUE(I)-ITRUE(J))  36,36,34
 0030               34 ITEM1 = ITRUE(I)
 0031                  ITEM2 = ITDEL(I)
 0032                  ITEM3 = IFIRST(I)
 0033                  ITRUE(I) = ITRUE(J)
 0034                  ITDEL(I) = ITDEL(J)
 0035                  IFIRST(I) = IFIRST(J)
 0036                  ITRUE(J) = ITEM1
 0037                  ITDEL(J) = ITEM2
 0038                  IFIRST(J) = ITEM3
 0039               36 CONTINUE
                C      TEST FOR ILLEGAL VALUES
 0040               40 IF (ITRUE(1)-1) 42,42,46
 0041               42 DO 44 J = 2,NTRUE
 0042                  ITRUE(J-1) = ITRUE(J)
```

```
          IFIRST(J-1) = IFIRST(J)
       44 ITDEL(J-1) = ITDEL(J)
          NTRUE = NTRUE-1
          GO TO 50
       46 IF (ITRUE(NTRUE)-NARRAY) 5C,48,48
       48 NTRUE = NTRUE-1
       50 DO 52 I = 2,NTRUE
          IF (ITRUE(I)-ITRUE(I-1)-1) 56,56,52
       52 CONTINUE
          GO TO 60
       56 NTRUE = NTRUE-1
          DO 58 J = I,NTRUE
          ITRUE(J) = ITRUE(J+1)
          IFIRST(J) = IFIRST(J+1)
       58 ITDEL(J) = ITDEL(J+1)
          GO TO 50
       60 GO TO 90
C         NO TRUE SITES INPUT, MAKE ONE UP
       70 ITRUE(1) = 10
          ITDEL(1) = 5
          IFIRST(1) = 20
          NTRUE = 1
C
C         PRINT ORIG SSA LOCATIONS AND DELTAS
       90 WRITE (3,91) (ITRUE(J),J=1,NTRUE)
       91 FORMAT (//10X,'ORIGINAL TRUE SITES, DELTAS, AND FIRST SCAN'//
        110X,10I5)
          WRITE (3,93) (ITDEL(J),J=1,NTRUE)
          WRITE (3,93) (IFIRST(J),J=1,NTRUE)
       93 FORMAT (/10X,10I5)
C
C         CHECK FOR INPUT OF EXTRA SITES
       94 IF (NDRAW) 99,99,9500
     9500 NCARDS = 16*NDRAW
          READ (1,9501) (ISCAN(J),J=1,NCARDS)
     9501 FORMAT (16I5)
          READ (1,9501) (IELEM(J),J=1,NCARDS)
          WRITE (3,9503)
     9503 FORMAT (///10X,'EXTRA INPUT SITES'/)
     9505 DO 9507 I = 1,NDRAW
          J1 = 16*(I-1)+1
          J2 = 16*I
          WRITE (3,9506) (ISCAN(J),J=J1,J2),(IELEM(J),J=J1,J2)
     9506 FORMAT (/12X,'SCAN    ',16I6/12X,'ELEMENT ',16I6)
     9507 CONTINUE
       99 CONTINUE
          WRITE (3,92)
       92 FORMAT (///20X,'CONTINUITIES'/12X,'LAST 2        LAST      TOTAL'/
        111X,'ELEMENTS       SCAN    OCCURANCES'//)
C
C         SET UP SCAN LOOPS
      100 NCONT = 0
          NPOSS = 0
          DO 225 KSCAN = 1,NSCAN
```

```
C      BACKGROUND FROM RANDOM NUMBERS
   115 CONTINUE
   116 DO 120 I=1,NARRAY
       CALL RANDU(IX,IY,YFL)
       IX = IY
       IARRAY(I) = 0
       YFL = YFL*100.
       IF (YFL-PBACK) 118,118,12C
   118 IARRAY(I) = 1
   120 CONTINUE
C
   122 IF (KSCAN-1) 125,125,130
C      OVERLAY TRUE SSA LOCATIONS ONTO BACKGROUND
   125 DO 128 K = 1,NTRUE
       IF (IFIRST(K)-KSCAN) 126,126-128
   126 J = ITRUE(K)
       IARRAY(J) = 1
   128 CONTINUE
       GO TO 148
C
C      AFTER FIRST SCAN, MODIFY SITES WHEN OVERLAYING
   130 DO 146 K = 1,NTRUE
       IF (IFIRST(K)-KSCAN) 132,132,146
   132 J = ITRUE(K)
       JD = ITDEL(K)
       JJD = J + JD
       IF (JJD-1) 146,146,133
   133 IF (JJD-NAR1) 135,146,146
   135 IARRAY(JJD) = 1
       ITRUE(K) = JJD
   137 CALL RANDU (IX,IY,YFL)
       IX = IY
       IF (YFL-.25) 138,138,139
   138 ITDEL(K) = JD-1
       GO TO 146
   139 IF (YFL-.95) 146,140,140
   140 IARRAY(JJD) = 0
   146 CONTINUE
   148 CONTINUE
C
C      CHECK FOR EXTRA LINE
   149 IF (NDRAW) 150,150,1491
  1491 DO 1492 J = 1,NCARDS
       IF (KSCAN-ISCAN(J)) 1492.1493,1492
  1492 CONTINUE
       GO TO 150
  1493 JJ = IELEM(J)
       IARRAY(JJ) = 1
   150 CONTINUE
C
C      BEGIN SEARCH FOR SINGLE SITES
   170 NSITES = 0
   172 DO 20C I = 2,NAR1
   173 IF (IARRAY(I)-1) 200,174,200
```

```
      C      CHECK FOR SINGLE
        174 IF (IARRAY(I-1)+IARRAY(I+1)) 185,175,185
        175 NSITES = NSITES + 1
            JSD = 1
            GO TO 1900
      C      CHECK FOR DOUBLE
        185 IF (IARRAY(I)+IARRAY(I+1)-2) 200,186,200
        186 IF (IARRAY(I-1)+IARRAY(I+2)) 200,192,200
      C      HAVE DOUBLE
        192 NSITES = NSITES + 1
            JSD = 2
      C      CONTINUITY BOOKKEEPING
       1900 CONTINUE
       1910 JSSA(NSITES,KSCAN) = I
       1911 IF (KSCAN-1) 1975,1975,1912
       1912 IF (KSCAN-2) 1950,1950,1915
       1915 IF (NCONT) 1950,1950,1920
      C      CHECK FOR SMALL WINDOW CONTINUITIES
       1920 NC = NCONT
            DO 1940 KCONT = 1,NC
            IF (JCONT(3,KCONT)-KSCAN) 1922,1940,1940
       1922 MULT = KSCAN-JCONT(3,KCONT)
            IDELTA = (JCONT(2,KCONT)-JCONT(1,KCONT))/JCONT(6,KCONT)
            IEXP = IDELTA*MULT + JCONT(2,KCONT)
       1923 IF (IABS(IDELTA)-1) 1924,1924,1925
       1924 IWSMA = IWSM1/2
            IWSMB = IWSM2/2
            GO TO 1928
       1925 IF (MULT-1) 1926,1926,1927
       1926 IWSMA = IWSM1
            IWSMB = IWSM2
            GO TO 1928
       1927 IWSMA = 1.5*IWSM1
            IWSMB = 1.5*IWSM2
       1928 IF (I-(IEXP-IWSMA)) 1940,1929,1929
       1929 IF (I-(IEXP+IWSMB)) 1932,1932,1940
      C      FITS INTO SMALL WINDOW, RECORD
       1932 NCONT = NCONT + 1
            JCONT(1,NCONT) = JCONT(2,KCONT)
            JCONT(2,NCONT) = I
            JCONT(3,NCONT) = KSCAN
            JCONT(4,NCONT) = JCONT(4,KCONT) + 1
            JCONT(5,NCONT) = 0
            JCONT(6,NCONT) = MULT
            JCONT(5,KCONT) = 1
       1940 CONTINUE
      C      CHECK BIG WINDOW POSSIBILITIES
       1950 CONTINUE
            IF (NPOSS) 1975,1975,1955
       1955 DO 1970 KPOSS = 1,NPOSS
            IF (JPOSS(2,KPOSS)-KSCAN) 1960,1970,1970
       1960 IEXP = JPOSS(1,KPOSS)
       1962 IF (I-(IEXP-IWBG1)) 1970,1963,1963
       1963 IF (I-(IEXP+IWBG2)) 1965,1965,1970
```

```
      C       FITS INTO BIG WINDOW
       1965 NCONT = NCONT + 1
            JCONT(1,NCONT) = JPOSS(1,KPOSS)
            JCONT(2,NCONT) = I
            JCONT(3,NCONT) = KSCAN
            JCONT(4,NCONT) = 2
            JCONT(5,NCONT) = 0
            JCONT(6,NCONT) = 1
       1970 CONTINUE
      C       ADD SS TO FUTURE POSSIBLES
       1975 NPOSS = NPOSS + 1
            JPOSS(1,NPOSS) = I
            JPOSS(2,NPOSS) = KSCAN
      C       END OF I ELEMENT LOOP
        200 CONTINUE
        202 NSSA(KSCAN) = NSITES
      C
      C       FINISH SCAN
      C       SORT CONTINUITIES
        204 IF (NCONT-1) 220,214,205
        205 NCONT1 = NCONT-1
        206 DO 212 IJ = 1,NCONT1
            IJ1 = IJ + 1
        208 DO 212 IK = IJ1,NCONT
            IF (JCONT(2,IJ)-JCONT(2,IK)) 212,209,210
        209 IF (JCONT(1,IJ)-JCONT(1,IK)) 212,212,210
        210 ITEMP1 = JCONT(1,IJ)
            ITEMP2 = JCONT(2,IJ)
            ITEMP3 = JCONT(3,IJ)
            ITEMP4 = JCONT(4,IJ)
            ITEMP5 = JCONT(5,IJ)
            ITEMP6 = JCONT(6,IJ)
            JCONT(1,IJ) = JCONT(1,IK)
            JCONT(2,IJ) = JCONT(2,IK)
            JCONT(3,IJ) = JCONT(3,IK)
            JCONT(4,IJ) = JCONT(4,IK)
            JCONT(5,IJ) = JCONT(5,IK)
            JCONT(6,IJ) = JCONT(6,IK)
            JCONT(1,IK) = ITEMP1
            JCONT(2,IK) = ITEMP2
            JCONT(3,IK) = ITEMP3
            JCONT(4,IK) = ITEMP4
            JCONT(5,IK) = ITEMP5
            JCONT(6,IK) = ITEMP6
        212 CONTINUE
      C       ELIMINATE ANY NON-CURRENT CR DUPLICATE ENTRIES
        214 I = 1
        215 CONTINUE
            IF (JCONT(5,I)) 2149,2149,2170
       2149 IF (KSCAN-JCONT(3,I)-2) 2151,2151,216
       2151 IF (I-NCONT) 2152,220,220
       2152 IF (JCONT(1,I)-JCONT(1,I+1)) 2153,2150,2153
       2150 IF (JCONT(2,I)-JCONT(2,I+1)) 2153,2154,2153
       2153 I = I+1
```

```
              GO TO 215
       2154   JCONT(4,I) = MAX0(JCONT(4,I),JCONT(4,I+1))
              JCONT(3,I) = MAX0(JCONT(3,I),JCONT(3,I+1))
              JCONT(5,I) = 0
              JCONT(6,I) = MAX0(JCONT(6,I),JCONT(6,I+1))
       2155   NCONT = NCONT-1
              IF ((I+1)-NCONT) 2156,2156,220
       2156   DO 2157 J = I,NCONT
              JCONT(1,J+1) = JCONT(1,J+2)
              JCONT(2,J+1) = JCONT(2,J+2)
              JCONT(3,J+1) = JCONT(3,J+2)
              JCONT(4,J+1) = JCONT(4,J+2)
              JCONT(5,J+1) = JCONT(5,J+2)
       2157   JCONT(6,J+1) = JCONT(6,J+2)
              GO TO 2152
     C        PRINT DISCARDS IF MORE THAN CERTAIN NUMBER OF SCANS
        216   IF (JCONT(4,I)-NCNTPR) 2170,2170,2160
       2160   WRITE (3,217) (JCONT(JPR,I),JPR=1,4)
        217   FORMAT (I14,I4,I10,I11)
       2170   NCONT = NCONT-1
              IF (I-NCONT) 2171,2171,220
       2171   DO 2172 J = I,NCONT
              JCONT(1,J) = JCONT(1,J+1)
              JCONT(2,J) =JCONT(2,J+1)
              JCONT(3,J) = JCONT(3,J+1)
              JCONT(4,J) = JCONT(4,J+1)
              JCONT(6,J) = JCONT(6,J+1)
       2172   JCONT(5,J) = JCONT(5,J+1)
              GO TO 215
        220   CONTINUE
     C        KEEP ONLY LAST SCAN POSSIBLES
        221   IF (NPOSS-1) 2240,2210,2215
       2210   IF (JPOSS(2,1)-KSCAN) 2211,2240,2240
       2211   NPOSS = 0
              GO TO 2240
       2215   DO 2216 IJ = 1,NPOSS
              IF (JPOSS(2,IJ)-KSCAN) 2216,2225,2225
       2216   CONTINUE
              NPOSS = 0
              GO TO 2240
       2225   IJ = IJ-1
              NPOSS = NPOSS - IJ
              IF (NPOSS) 2240,2240,2226
       2226   DO 2230 IK = 1,NPOSS
              IJK = IJ + IK
              JPOSS(1,IK) = JPOSS(1,IJK)
       2230   JPOSS(2,IK) = JPOSS(2,IJK)
       2240   CONTINUE
     C
     C        END OF SCAN LOOP
        225   CONTINUE
     C
     C        PRINT REMAINING CONTINUITIES IF MORE THAN INPUT QTY
              IF (NCONT) 899,899,230
```

```
0275         230 DO 235 K = 1,NCONT
0276             IF (JCONT(4,K)-NCNTPR) 235,235,232
0277         232 WRITE (3,217) (JCONT(JPR,K),JPR=1,4)
0278         235 CONTINUE
           C
0279         899 IF (NARRPR) 9CC,900,940
           C     PRINT OUT ALL SSA LOCATIONS
0280         900 WRITE (3,901)
0281         901 FORMAT(1H1//10X,13HSSA LOCATIONS///16X,12HELE     SCANS/
                115X,5HMENTS,9X,1H1,9X,1H2,9X,1H3,9X,1H4,9X,1H5,9X,1H6,9X,1H7,
                29X,1H8,9X,1H9/
                320X,100H1234567890123456789012345678901234567890123456789012345678
                4901234567890123456789012345678901234567890//)
           C     SET UP A LINE FOR EACH ELEMENT
0282         908 DO 909  I = 1,NSCAN
0283         909 NPLACE(I) = 1
0284         910 DO 935 I = 1,NARRAY
           C     LINE TO SPACES
0285             DO 912 J = 1,100
0286         912 PLINE(J) = SPACE
           C     SEARCH WITHIN EACH SCAN
0287         914 DO 922 KSCAN = 1,NSCAN
0288             K = NPLACE(KSCAN)
0289             IF (JSSA(K,KSCAN)-I) 922,918,922
0290         918 PLINE(KSCAN) = STAR
0291             NPLACE(KSCAN) = NPLACE(KSCAN) + 1
0292         922 CONTINUE
           C     PRINT LINE
0293         930 WRITE (3,931) I,(PLINE(J), J = 1,NSCAN)
0294         931 FORMAT (I18,2X,100A1)
0295         935 CONTINUE
           C
0296         940 CONTINUE
0297             GO TO 10
0298         999 CONTINUE
0299             CALL EXIT
0300             END
```